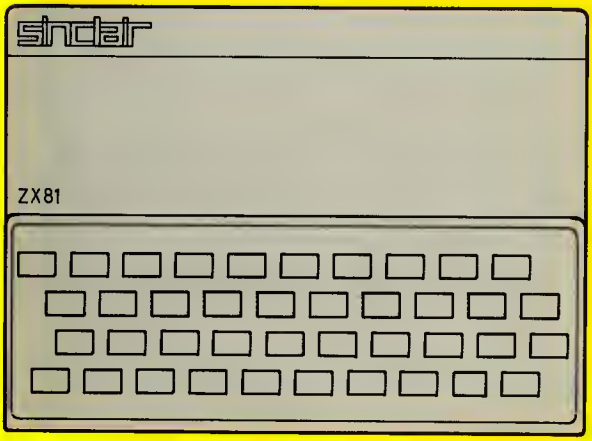
ZX 81

LEREN PROGRAMMEREN

16K
16K
16K
16K
16K
16K
16K
16K



M. JAMES - S. M. GEE

DE MUIDERKRING

ZX 81
16K

CIP-GEGEVENS

James, M.

ZX81:16K: Leren programmeren/M. James, S.M. Gee;
(vert. uit het Engels door Jos Verstraten). - Bussum: De Muiderkring - ill.
Vert. van: The art of programming the 16K ZX81. - London: Babani, 1982.
ISBN 90-6082-248-X
SISO 365.3 UDC 681.31.3.06
Trefw.: programmeren; microcomputers.

ISBN 90 6082 248 X

M. JAMES - S. M. GEE

ZX 81

16K

LEREN PROGRAMMEREN

DE MUIDERKRING B.V. – BUSSUM
UITGEVERIJ VAN TECHNISCHE BOEKEN EN TIJDSCHRIFTEN

INHOUD

VOORWOORD

Dit boek is voornamelijk bedoeld voor bezitters van de Sinclair ZX81 computer in basisuitvoering (dus met 1K aan geheugen) die een beetje zijn uitgekeken op de beperktheid van hun apparaat en van plan zijn een 16K-uitbreidingsunit en misschien zelfs de ZX-printer aan te schaffen. De bedoeling is u de juiste stijl van programmeren te leren en u vertrouwd te maken met de capaciteiten van de 16K-RAM-unit en van de ZX-printer.

In hoofdstuk 1 wordt de nieuwe "hardware" gepresenteerd: de Sinclair 16K-geheugenuitbreiding en de printer van hetzelfde merk. In hoofdstuk 2 wordt uitgelegd hoe we de nieuwe geheugenruimte kunnen gebruiken en wordt een simpel programma beschreven, waarmee men de werking van de geheugenunit kan controleren.

Het schrijven van grotere programma's vereist bepaalde technieken, deze komen in hoofdstuk 3 aan de orde.

Hoofdstuk 4 is het voorspel tot het betere werk: door middel van vijf uitgewerkte programma's voor populaire spelletjes wordt de gebruiker gewezen op de vermeerderde grafische mogelijkheden van de 16K-computer. De vier volgende hoofdstukken gaan over het schrijven en het corrigeren van grotere programma's, de technieken om deze programma's op cassette op te nemen en zowel het programma als het resultaat door middel van de ZX-printer te vereeuwigen. In deze hoofdstukken komen ook de technieken aan bod die nodig zijn voor het opstarten van gegevensbestanden, het uitwerken van statistische analyses, het opzetten van eenvoudige boekhoudsystemen en het uitprinten van tekst en figuren. In hoofdstuk 9 wordt dieper ingegaan op het werken met willekeurige getallen. Het laatste hoofdstuk, tenslotte, introduceert het programmeren in machinecode en geeft duidelijk aan wat de voordelen van deze techniek zijn.

Als u, met dit boekje als gids, er achter komt hoe veelzijdig en krachtig uw "nieuwe" 16K ZX81 is geworden, dan zijn wij in onze opzet geslaagd.

M. James; S. M. Gee

1. BREID UW ZX 81 UIT!

Het programmeren op de ZX81 met slechts 1K aan geheugencapaciteit mag voor de beginnende programmeur spannend en uitdagend zijn, na niet al te lange tijd slaat die spanning echter gegarandeerd om in een grote frustratie. Al programmerend en lerend ontdekt men de onvoorstelbaar vele mogelijkheden van de ZX81, maar ondervindt tegelijkertijd hoe weinig met de basisuitvoering van de machine haalbaar is. Een aantal van de unieke eigenschappen van deze machine, zoals bijvoorbeeld het tekenen van bewegende beelden, komen helemaal niet tot zijn recht met slechts 1K aan geheugen. Eenmaal de fundamenten van de ZX-BASIC-taal onder de knie hebbend, ontstaat al snel de behoefte eigen ideeën over programma's te willen uittesten en dan jeuken de handen om de capaciteit van de machine te vergroten. Gelukkig dat dit bij de ZX81 zeer gemakkelijk kan; er staat op dit moment een keur aan uitbreidingsunits ter beschikking.
Over twee van deze uitbreidings-"hard-ware" gaat dit boekje: de 16K-geheugenunit en de ZX-printer.

De 16K-RAM-unit
Er zijn op dit moment diverse geheugenunits voor de ZX81 in de handel. Hoewel deze allemaal voldoen en in grote lijnen op dezelfde manier werken, hebben wij dit boekje geschreven aan de hand van de door Sinclair Research zélf op de markt gebrachte "16K-RAM-pack". Deze kompakte unit bevat acht 2K-RAM geheugen-IC's en is voorzien van een tweezijdige printplaatconnector waarmee hij met de print van de ZX81 kan worden verbonden. Nu is deze verbinding vrij kritisch, want een onderbreking, al is het maar voor even, betekent onherroepelijk het verlies van alle, vaak met veel moeite, in het geheugen geschreven informatie! En daar het in de praktijk wel eens wil voorkomen, dat er tegen de machine wordt gestoten, waardoor de verbinding tussen ZX81 en geheugenmodule wordt onderbroken, is het toch wel de moeite waard even bij dit zwakke punt te blijven stilstaan.

Er zijn een aantal remedies te verzinnen, waarvan de meest radikale is het vastsolderen van de unit aan de "moeder"-print van de computer. Hoewel dit zonder meer de beste oplossing is, raden wij hem toch af, omdat u dan nooit meer iets anders op de computer kunt aansluiten.
Niet fraai, maar wel praktisch is het volspuiten van de ruimte tussen de achterwand van de computer en de unit met plastisch blijvende siliconenkit. Deze versteviging voldoet voor het opvangen van alle normale mechanische belastingen van de machine, zoals stoten en verschuiven. Bovendien kan deze kit in voorkomende gevallen met een scherp mesje worden losgesneden, waardoor de expansie-"bus" op de achterzijde van de computer weer ter beschikking staat voor andere uitbreidings-"hardware". Dat is bijvoorbeeld al nodig als we, naast de 16K-unit, ook de ZX-printer willen gaan gebruiken. De printer-interface, het kastje dat de extra elektronica bevat voor het sturen van de printer, moet dan in de plaats van de 16K-unit rechtstreeks in de ZX81 worden geplugd. De RAM-pack past dan op haar beurt in de achterzijde van de printer-interface. Het zal duidelijk zijn, dat de totale konstruktie nu extra gevoelig is voor stoten; de weg tussen "moeder"-print en geheugenuitbreiding is op deze manier extra lang! Toch is gebleken dat er nauwelijks contactproblemen te verwachten zijn, als de combinatie computer-interface-RAM-pack op een gladde, harde ondergrond wordt opgesteld.
U kunt er op rekenen, dat alles goed zit als de verbinding tussen computer en RAM-pack niet buigt of beweegt, als enige toetsen van het toetsenbord worden aangeraakt, zoals dat bij het intikken van programma's gebeurt.
De geringste beweging hierbij verraadt een niet stabiele opstelling! U kunt dan proberen het geheel met tweezijdig klevend plakband op een stevige en gladde ondergrond, zoals bijvoorbeeld een stuk meubelplaat, te verankeren.

Nieuwe programmeervooruitzichten!
Het beschikken over 16K extra geheugenruimte maakt het leven van een programmeur veel ge-

makkelijker. De noodzaak om listige en vergezochte truukjes te moeten toepassen om programma's korter te maken, vervalt immers.
Zo ook de ergernis, vanwege plaatsgebrek, extra's niet in programma's te kunnen opnemen.
De nadruk kan nu liggen waar hij moet liggen: bij het rustig schrijven van goed gestileerde programma's.
U zult ervaren dat het gebruik van de ZX81 veel leuker wordt. U kunt nu immers veel ingewikkelder programma's aanpakken en meer uiteenlopende problemen op uw computer uitwerken. Maar al doende, zult u ondervinden dat het uitbuiten van alle nieuwe mogelijkheden ook meer kennis vereist. Kennis over nieuwe programmeertechnieken en andere inzichten over het omgaan met het geheugen. Vergeet niet dat u opeens de beschikking heeft over niet minder dan 16 keer de originele geheugenruimte!
Uw "oude" ZX81 kon al heel wat, maar met de "nieuwe" kan met aangepaste programmeertechnieken nog veel meer te voorschijn worden getoverd. Hierop is dit boek gebaseerd. Het leert u deze nieuwe technieken beheersen, begrijpen en gebruiken om uw 16K-geheugenruimte uit te buiten. Daarnaast draagt het voldoende materiaal aan om zelf aan de slag te gaan.

De ZX-printer

U moet echt gaan denken aan het aanschaffen van een ZX-printer! Nu u over zoveel extra geheugen kunt beschikken, en programma's kunt uitschrijven, die niet meer in één keer op het schermbeeld passen, kunt u in feite niet buiten een printer. Het is zo goed als onmogelijk fouten in deze programma's op te sporen door het scherm af te lezen. Vaak moet u immers door "GO TO"-instrukties op het einde terug naar het begin en dat is alleen met een volledig uitgeschreven programma nog enigszins te overzien.
U kunt uiteraard uw programma's met de hand uitschrijven; waarschijnlijk ontstaan er dan nog meer fouten!
Kortom, de enige praktische methode om lange programma's te korrigeren, is dit met de ZX-printer te doen. Bovendien betekent het gebruik van de printer tal van nieuwe toepassingen voor uw computer.
U kunt gegevens, zoals financiële overzichten en adressenbestanden, bewaren op een papierstrook, de zogenoemde "hard copy". De ZX-printer kan echter meer dan enkel letters en cijfers uitschrijven. Met de 16K-unit kunt u immers experimenteren met grafische voorstellingen met hoge resolutie.
Ook deze grafische voorstellingen kunnen door de printer op papier worden vastgelegd en bewaard voor later.
Natuurlijk heeft de ZX-printer zijn beperkingen.
Vergeleken met andere op de markt zijnde apparaten, is de maximale schrijfbreedte zeer gering en hij gebruikt glanzend, grijsachtig aluminium-papier, dat er niet zo leuk uitziet. In de praktijk blijkt echter dat een ZX-printer-output zonder problemen kan worden gefotokopiëerd. De kopie is erg scherp en mooi zwart op wit.
Een groot bezwaar is het flikkeren en rollen van het beeld, als men de printer de beeldinformatie laat uitplotten. Bij het schrijven op papier maakt hij nogal wat lawaai en veroorzaakt storingen op uw beeldscherm.
Zijn enige, echte voordeel is zijn onvoorstelbaar lage prijs, vergeleken met de printers die er op dit moment op de markt zijn.
De prijs-prestatie-verhouding ligt bij de ZX-printer zeer hoog!

Hoe u dit boek moet gebruiken

Dit boek veronderstelt een bepaalde kennis van de ZX81 en de BASIC-programmeertechnieken. Wij gaan er vanuit, dat de lezer enige ervaring met de 1K-uitvoering van de computer heeft opgedaan en dat het niet noodzakelijk is alle in dit boek toegepaste technieken uit te leggen.
Verder gaan we in op het serieuze werk: het opstellen van nuttige programma's op het gebied van statistiek, financieën beheer- en gegevensbestanden.
Uw ZX81 is immers door het uitbreiden van zijn geheugencapaciteit van een eenvoudig speeltuigje bevorderd tot een "echte" microcomputer.
Daarom kunt u ook een meer wetenschappelijke aanpak van de programmeertechnieken verwachten.
Het kan best zijn, dat u bij het lezen van de volgende hoofdstukken af en toe met uw ogen zult knipperen, want soms gaan we er tamelijk hard tegenaan!

Als u het allemaal niet zo één-twee-drie kunt volgen, is er geen man over boord. U kunt de programma's gewoon gebruiken en er later op terugkomen als u meer inzicht heeft gekregen door het zelf schrijven van langere programma's.
Het is niet noodzakelijk het gehele boekje door te ploeteren. Als u bijvoorbeeld voornamelijk in het programmeren van spelletjes bent geïnteresseerd, kunt u eenvoudig verder gaan met hoofdstuk 4. Net zo goed kunt u meteen naar hoofdstuk 7 "springen" als uw toepassingen zich op het gebied van de statistiek afspelen.
Wat u ook met dit boekje van plan bent, bekijk wél hoofstuk 3 waar een aantal zeer nuttige "utilities" (kleine hulpprogramma's die het opstellen van grotere programma's veraangenamen) worden gegeven!

2. 16K EXTRA GEHEUGENRUIMTE!

Zoals u wellicht reeds weet, kunnen we in 1K aan geheugenruimte precies 1024 symbolen onderbrengen. Met een beetje rekenwerk komen we er dan achter, dat we in 16K aan geheugenruimte niet minder dan 16384 symbolen kunnen opslaan. Dit lijkt ontzettend veel en het ligt voor de hand ons af te vragen of we deze immense capaciteit ooit nodig zullen hebben.
Het antwoord is simpel: ja!
Ondanks de geheugencapaciteit van 16384 plaatsen kan het in de praktijk voorkomen dat een programma alle geheugenplaatsen vult.
Zo zal het onderstaande één-regelige programma:

```
10 DIM A(50,100)
```

een "out of memory" foutindicatie (code 4) oproepen. Hoewel het een heidens karwei is om met de hand het geheugen vol te schrijven met 16384 symholen, gebruiken programma's soms voor ogenschijnlijk de eenvoudigste dingen grote geheugengebieden.

Geheugenkaart en systeemvariabelen
In het algemeen wordt de beschikbare geheugenruimte van de ZX81 voor vier verschillende zaken gebruikt:

1) het opslaan van programma's;
2) het opslaan van gegevens;
3) het opslaan van schermbeeldinformatie;
4) het opslaan van systeemgegevens.

De meest voor de hand liggende taak van het geheugen is het opslaan van programma's. Iedere programmaregel die u via het toetsenbord in de computer voert, wordt opgeslagen in het geheugen. In het algemeen kan men stellen dat iedere regel evenveel bytes in het geheugen in beslag neemt als er toetsaanslagen nodig zijn voor het schrijven van de regel.
De tweede voorname taak van het geheugen is het opslaan van gegevens. Iedere door u gebruikte variabele krijgt (door hem een naam te geven in het programma) een bepaalde ruimte in het geheugen toegewezen, waarin zijn waarde kan worden bewaard. Een gemiddelde variabele eist ca 6 bytes op, er schijnt dus in de totaal beschikhare 16K genoeg plaats te zijn voor een flink aantal variabelen. Dat gaat op, tot men met array's (gebieden) te maken krijgt! Deze verslinden een grote hoeveelheid geheugen en helaas is het werken met variabelen array's nu net één van de kenmerken van het geavanceerde programmeren.
De twee laatstgenoemde taken van het geheugen liggen niet zo voor de hand en onttrekken zich voor het grootste gedeelte aan de controle van de gebruiker. De ZX81 reserveert een geheugendeel voor het opwekken van het schermbeeld.
Hoe dat precies in zijn werk gaat, is onder meer afhankelijk van de beschikbare geheugenruimte. Bij de 16K ZX81 worden standaard 793 bytes voor de schermbeeldinformatie gebruikt, onafhankelijk van wat er op het scherm te zien is. In eerste instantie lijkt het een beetje vreemd zo veel geheugen vrij te houden voor het weergeven van bijvoorbeeld een blanco scherm, maar de voordelen van deze standaard-procedure wegen ruimschoots op tegen het verlies aan geheugenruimte. Dit komt in hoofdstuk 4 aan de orde.

Naast deze geheugenruimte voor het beeld, heeft de machine ruimte nodig om interne gegevens op te slaan. Interne gegevens zijn bijvoorbeeld informatie over waar een bepaald programma eindigt of over de plaats van de cursor op het scherm. Dit soort geheugenplaatsen worden "systeem-variabelen" genoemd en er zijn bij alle ZX81-uitvoeringen 124 bytes voor deze variabelen gereserveerd.
Het kan nodig zijn te weten hoe alle genoemde taken in het totale geheugen zijn ondergebracht. Vandaar dat een "geheugenkaartje" (memory-map) op zijn plaats is.

adres	gebruik
16384	systeem-variabelen
16509	programma
D-FILE	display-file
VARS	variabelen
E-LINE	regel, die wordt geschreven
STKBOT	calculator-stack
STKEND	vrije RAM-ruimte
stack-pointer	machine-stack
ERR-SP	gosub-stack
RAMTOP	vrije RAM-ruimte
32767	

Uit dit overzicht blijkt, dat sommige indelingen van het geheugen beginnen en eindigen op vaste adressen en andere op variabele.
Zo zal het programmagebied altijd starten op 16509, maar het daaropvolgende gebied (display-file) begint op een lokatie, die afhankelijk is van de lengte van het programma.
De symbooladressen liggen op vaste plaatsen in het "systeem-variabelen"-gebied van het geheugen, zodat de ZX81 ze steeds zeer snel kan terugvinden. Een volledige lijst van deze (en andere) vaste plaatsen vindt u in hoofdstuk 28 van de ZX81-handleiding.
Het gebruik van dit soort geheugenlokaties is eenvoudig, maar u moet wél eerst wennen aan het feit dat twee geheugenlokaties worden gebruikt voor het opslaan van het adres van één andere geheugenlokatie! Zo wordt de systeemvariabele "D-FILE" opgehorgen op de geheugenplaatsen 16396 en 16397. In deze twee lokaties bevindt zich het adres van de geheugenplaats waarin het begin van de beeldinformatie is opgeslagen.
U moet wel voor ogen houden, dat BASIC een begrip als "D-FILE" niet verstaat. Wilt u toegang tot deze informatie, dan moet u met adressen werken.

Het manipuleren met geheugenlokaties
"PEEK" en "POKE" zijn de twee BASIC-instrukties, waarmee we toegang krijgen tot de geheugenlokaties. De funktie:

```
PEEK ADRES
```

roept de inhoud van de geheugenplaats met het betreffende adresnummer op, en

```
PEEK ADRES,WAARDE
```

slaat de gegeven "waarde" op in de geheugenplaats die is gedefinieerd door het gegeven adresnummer.
In één geheugenlokatie kunnen getallen tussen 0 en 255 worden opgeslagen. Het adres van een ZX81 geheugenplaats kan liggen tussen 0 en 65534! Om dus één adres te kunnen opslaan, hebben we twee geheugenplaatsen nodig.
We hoeven ons niet te verdiepen in de vrij ingewikkelde binaire rekenwijze, waarmee de computer dit soort opdrachten uitvoert. Dat maakt het programmeren maar nodeloos gecompliceerd! Hoe alles in zijn werk gaat, kunnen we net zo goed verklaren aan de hand van de bekende decimale getallenindeling, die ook in programmaregels wordt gebruikt.
Zoals reeds gezegd, kan één geheugenplaats (of byte) getallen tot en met 255 opslaan. In één enkele geheugenplaats kunnen we dus tellen van 0 tot en met 255. Het eerstvolgende getal 256 past niet meer in die plaats, dus moeten we de tweede geheugenlokatie te hulp roepen. Door daar een 1 in te zetten, geven we aan dat de eerste geheugenplaats één maal is volgeteld. Nadien kunnen we rustig verder gaan met voor de tweede maal de eerste geheugenplaats vol te tellen (startend vanaf 0). De tweede geheugenplaats wordt dus gebruikt om te tellen hoe vaak de eerste is volgeteld met 256 getallen. De eerste geheugenplaats, die de eenheden telt, wordt "least significant byte, LSB" genoemd, de minst belangrijke dus. De tweede geheugenplaats, die veelvouden van 256 telt, wordt "most significant byte, MSB" ofte wel de meest belangrijke genoemd.

eerste geheugenplaats telt eenheden	tweede geheugenplaats telt veelvouden van 256
LSB	MSB

Het zal nu duidelijk zijn, hoe we het adres kunnen reconstrueren uit de inhoud van twee geheugenlokaties. Door middel van een "PEEK" kunnen we de inhoud van de eerste oproepen, nadien tellen we hij deze waarde de met 256 vermenigvuldigde inhoud van de tweede op. In de vorm van een BASIC-bevel:

```
PEEK M+256*PEEK(M+1)
```

Dit bevel roept de inhoud op van de twee geheugenlokaties, opgeslagen in de geheugenplaatsen met de adressen "M" en "M + 1".
De omgekeerde bewerking gaat al even gemakkelijk. Als u een bepaald getal in twee delen moet splitsen, omdat het langer is dan één byte (geheugenplaats), dan moet u eerst dat getal delen door 256 en het resultaat van die deling in de MSB opslaan. De rest van de deling hoort thuis in de LSB. In de vorm van een BASIC-bevel:

```
POKE M+1,INT(V/256)
POKE M,V-256*INT(V/256)
```

Dit kommando zal het getal "V" opbergen in de geheugenplaatsen "M" en "M + 1".
Zowel "PEEK" als "POKE" zullen we in de toekomst vaak toepassen.

De overige geheugenplaatsen

De diskussie over het adresseren van geheugenplaatsen heeft ons even afgeleid van wat we aan het bespreken waren: de indeling van het geheugen, aan de hand van de "memory map".
De "calculator-stack" is het geheugengebied, waarin de machine tijdelijk de resultaten van berekeningen opslaat. De omvang van deze zone varieert zelfs tijdens het uitwerken van een programma en is afhankelijk van de soort berekening.
De "machine-stack" heeft als bijzonderheid, dat niet is te achterhalen waar dit deel van het geheugen eindigt. Deze informatie wordt namelijk bewaard in de "machine stack pointer" en is met behulp van BASIC niet toegankelijk. Dat is niet zo erg, want in dit deel van het geheugen wordt niets bewaard wat voor het programmeren van belang is.
De "GOSUB-stack" wordt gebruikt voor het opslaan van het nummer van de programmaregel, waarmee we via een "RETURN"-bevel uit een subroutine terugkeren naar het hoofdprogramma.
Het laatste deel van het geheugen is de zogenoemde "free RAM", met andere woorden, de vrije RAM-plaatsen, die ter beschikking van de programmeur staan. De "systeem-variabele" "RAMTOP" wordt ingevuld op het adres van de hoogste RAM-lokatie, bij het inschakelen van de machine. Toch is het mogelijk de waarde van de "RAMTOP" te veranderen, als we bijvoorbeeld meer geheugenruimte voor speciale toepassingen willen vrijhouden. Dat kunnen bijvoorbeeld subroutines in machinecode zijn. Dit wordt in hoofdstuk 10 behandeld.

Een programma voor het testen van het geheugen

Een goed voorbeeld hoe we met de systeem-variabelen kunnen omgaan is het volgende programma voor het testen van de werking van het geheugen.
Het ligt voor de hand dat we de werking van een geheugenplaats kunnen kontroleren door er eerst "iets" in op te slaan, daarna dit gegeven uit te lezen en vervolgens te kijken of er verschillen zijn ontstaan tussen in- en uitgelezen informatie.
Het prohleem is dat "iets" zo te bepalen dat we er zeker van zijn dat het testprogramma werkt. Dat is niet zo eenvoudig, want defekte geheugenlokaties kunnen bepaalde informatie foutloos verwerken en weer vrijgeven, terwijl andere informatie niet of met fouten wordt verwerkt.

Een mogelijkheid is gebruik te maken van de toevalsfuncties van de ZX81, waarmee we een willekeurige en uitcenlopende reeks getallen kunnen opwekken. Het geheugentestprogramma moet dus een willekeurig getal opslaan in de eerste vrije RAM-lokatie, dit getal uitlezen en vergelijken met de ingelezen waarde.
Dit proces moet worden herhaald voor iedere plaats in de "free RAM"-zone van het geheugen. Het zal duidelijk zijn, dat we alleen maar de "free RAM"-zone kunnen testen. Het invoeren van willekeurige gegevens in andere delen van het geheugen zou namelijk het programma, dat immers ook ergens in het geheugen staat, beschadigen!
Het omzetten van het principe in een door de computer te begrijpen aantal BASIC-regels is niet zo moeilijk.

```
10 PRINT "+";
20 LET E=PEEK 16386+256*PEEK 16387 -40
30 LET M=PEEK 16412+256*PEEK 16413 +20
40 LET M=M+1
50 IF M>E THEN GOTO 10
60 LET R=INT(RND*256)
70 POKE M,R
80 IF PEEK(M)=R THEN GOTO 40
90 PRINT "FOUT IN ";M;" VERWACHT ";R
100 PRINT "GELEZEN ";PEEK M
110 GOTO 40
```

Het programma start met het printen van een "+"-symbool op het scherm (regel 10), zodat we weten dat het programma loopt. Nadien bepalen we door middel van de regels 20 en 30 waar de vrije RAM-zone zich bevindt. We vullen de variabele "E" in op het adres van de laatste vrije byte, door het "PEEK"-en van de lokaties "ERR-SP" (het begin van de "GOSUB-stack") en daarvan 40 af te trekken om voldoende ruimte te houden voor de "machine-stack". We weten immers dat we met BASIC nooit de juiste plaats van het einde van deze stack kunnen terugvinden!
Nadien vullen we de variabele grootheid "M" in op het adres van de eerste byte van de vrije RAM-zone, door het "PEEK"-en van de "STKEND"-lokaties (het einde van de calculator-stack) en daarbij 20 op te tellen. In principe lijkt het overbodig een extra ruimte van 20 bytes aan te houden, omdat "STKEND" exact het begin van de vrije zone aangeeft. De computer moet echter bij het uitvoeren van dit programma een aantal herekeningen maken, waardoor het kan gebeuren dat de "calculator-stack" breder wordt en dank zij deze 20 extra bytes kan dat zonder problemen.
De kommando's in de regels 40 tot en met 80 zorgen voor het achter elkaar testen van alle lokaties in de vrije zone. Regel 60 wekt een willekeurig getal op, hetgeen door regel 70 wordt opgeslagen in de byte, die wordt getest. Met regel 80 roepen we de informatie terug en vergelijken deze met de originele "R"-waarde. Als "PEEK (M)" gelijk is aan "R", wordt in regel 40 door middel van een "GOTO" de waarde van de variabele "M" met één eenheid verhoogd. Zou er een fout worden aangetroffen ("PEEK (M)" is dan niet gelijk aan "R"), dan wordt er een foutmelding geprint met behulp van de regels 90 en 100, waarna het testen gewoon doorgaat.
Als alle vrije ruimte is getest, wordt "M" gelijk aan "E" en print regel 50 een tweede plusje op het scherm.
Het testprogramma start vervolgens opnieuw en doorloopt voor de tweede keer alle vrije geheugenplaatsen.
Tijdens het uitdraaien van het programma zou u kunnen denken dat er iets fout is gegaan omdat er maar één plusteken op het scherm verschijnt. Toch hoeft dat niet te betekenen dat er iets niet in orde is, het duurt namelijk meer dan een half uur alvorens de computer alle vrije plaatsen heeft gekontroleerd!
Dit simpele voorbeeld maakt ons attent op een van de problemen bij het het schrijven van grote programma's op de ZX81: de snelheid!

3. PROGRAMMA-UTILITIES

Een "utility" (letterlijk vertaald een nuttigheidje) is in het algemeen een programma dat gebruikt kan worden bij het schrijven van andere programma's.
Zo zou u de in het vorige hoofdstuk beschreven geheugentest als een "utility" kunnen beschouwen, omdat het de machine controleert alvorens u programma's gaat intoetsen.

Nu we de beschikking hebben over 16K aan geheugen-ruimte en we ons geen zorgen hoeven te maken of onze programma's wel in het geheugen passen, kunnen we een deel van deze capaciteit reserveren voor het opslaan van "utilities", die goed van pas komen bij het intoetsen van grotere programma's.
Zo kan het tijdens het invoeren van programma's wel eens nodig zijn te weten hoeveel geheugen er nog vrij is en hoe de computer alle informatie in zijn geheugen heeft verwerkt. Met onze kennis van de geheugenindeling en de systeemvariabelen kunnen we zo'n programma schrijven, maar we zullen hiervoor tamelijk ingewikkelde technieken moeten toepassen. Het zou ons niet verbazen als u op dit moment de uitleg laat voor wat hij waard is en gewoon de "utility" in praktijk gaat brengen. Dit geldt trouwens voor alle in dit hoofdstuk beschreven hulpprogramma's.
Na meer praktijkervaring, kunt u altijd naar dit hoofdstuk teruggrijpen om de fijne kneepjes van de "utility"-programma's te ontdekken.

De drie in dit hoofdstuk beschreven hulpprogramma's zijn zo genummerd (in het 9000-gebied) dat ze steeds achter een programma kunnen worden opgenomen. Bovendien gebruiken we voor iedere "utility" een andere set regelnummers, zodat ze alle drie gezamenlijk kunnen worden ingevoerd en door middel van "GOTO"-s afzonderlijk kunnen worden opgeroepen. In hoofdstuk 5 wordt een methode beschreven, waarmee deze programma's zo kunnen worden georganiseerd, dat ze in een hoofdprogramma kunnen worden geïntegreerd.

"Utility" nummer 1: geheugengebruik
Deze utility, waarmee we het gebruik van het geheugen kunnen kontroleren, is een tamelijk kort hulpprogramma dat we praktisch bij ieder hoofdprogramma kunnen gebruiken:

```
9200 PRINT "PROGRAMMA ";PEEK 16396+256*PEEK
  16397-16509
9210 PRINT "BEELDSCHERM ";PEEK 16400+256*PEEK
  16401-PEEK 16396-256*PEEK 16397
9220 PRINT "VARIABELEN ";PEEK 16404+256*
  PEEK 16405-PEEK 16400-256*PEEK 16401
9230 PRINT "VRIJ GEHEUGEN ";PEEK 16386+256*PEEK
  16387-PEEK 16412-256*PEEK 16413
9240 STOP
```

U kunt dit programma het best vanaf audiotape in het geheugen laden, alvorens u het echte programma gaat schrijven. De nog vrije geheugenruimte kan op ieder gewenst moment zichtbaar worden gemaakt door regel "GOTO 9200" in te tikken. De hoeveelheid door een programma ingenomen geheugenruimte wordt bepaald door het verschil te berekenen tussen het adres van het begin van de programmaruimte, namelijk 16509 en het adres van de laatste byte van de programmazone, opgeslagen in "D-FILE" (16396 en 16397). Op dezelfde manier berekenen we de geheugenruimte, die in beslag wordt genomen door de variabele grootheden en door de schermbeeldinhoud.
Dit hulpprogramma kreëert geen nieuwe variabelen.

Wél vraagt deze "utility" wat ruimte in het programmadeel van het geheugen.
Het nog vrije deel van het geheugen wordt bepaald door het berekenen van het verschil tussen het adres, opgeslagen in "STKEND" en het adres in "ERR-SP". Dit systeem houdt dus geen rekening met de ruimte die door de "machine-stack" in beslag wordt genomen en het resultaat is bijgevolg 5 tot 10 plaatsen te hoog!

"Utility" nummer 2: het gebruik van variabelen

Hoewel we met "utility" nummer 1 kunnen berekenen hoeveel geheugenruimte er door de variabelen wordt bezet, hebben we met dat hulpprogramma nog geen overzicht van de verdeling van die geheugenruimte tussen de verschillende variabelen.

Vaak komt het bij het schrijven van zeer ingewikkelde programma's voor dat men vergeet welke variabelen er al in gebruik zijn. U loopt dus het risico dezelfde naam voor verschillende variabelen in te voeren!

Met de in deze paragraaf beschreven "utility" kunt u op ieder gewenst moment een volledig overzicht krijgen over alle reeds gebruikte variabelen.

Het programma bekijkt het variabelengebied van het geheugen en maakt een lijstje met de naam, soort en momentele waarde (zo die aanwezig is) van alle gevonden variabelen.

Het is, met alles wat we ondertussen aan de weet zijn gekomen over de indeling van het geheugen, niet zo moeilijk om start- en eindpunt van het variabelengebied terug te vinden. Veel ingewikkelder wordt het, als we de in het geheugen gevonden informatie op de beschreven manier willen gaan interpreteren. Daarvoor moeten we eerst dieper ingaan op de wijze waarop de computer variabelen opslaat in zijn geheugen.

De ZX81 herkent zes verschillende soorten variabelen, hoewel de menselijke gebruiker van het apparaat soms niet in staat is de verschillen te ontdekken. Ieder type variabele eist een eigen specifieke geheugenruimte op voor het opslaan van de waarde en overige relevante informatie. Wat ze echter allemaal met elkaar gemeen hebben, is dat de eerste geheugenplaats wordt gebruikt om het soort variabele te identificeren. Om ruimte uit te sparen wordt deze eerste geheugenlokatie ook gebruikt om de eerste en zo mogelijk enige, letter van de naam van de variabele in op te slaan. Een en ander gaat vrij eenvoudig.

Zoals we weten, moet een variabele grootheid door een **LETTER** worden voorgesteld. Omdat er slechts 26 letters bestaan, kunnen we de naam van de grootheid voorstellen door een getal tussen 1 en 26. Daarnaast onderscheidt de computer slechts zes soorten van variabelen, zodat ook deze indeling vrij simpel kan worden gecodeerd.

code	soort van variabele
2	string met willekeurige lengte
3	eenvoudige variabele (één-letterig)
4	nummer-array
5	langer-dan-een-letter variabele
6	symbolen-array
7	index-variabele, gebruikt in "FOR"-lus

Beide delen van de informatie (type en naam) hebben slechts een beperkte omvang, daarom kunnen ze zonder problemen in één byte worden ondergebracht.

Dat gaat als volgt: de drie meest belangrijke bits van de byte worden gebruikt voor het opslaan van de typecode en de vijf resterende minst belangrijke bits worden gebruikt voor het bewaren van de letter van de naam van de variabele.

Dus:

geheugenplaats
byte

bit	7	6	5	4	3	2	1	0
	T	T	T	L	L	L	L	L
	type			letter				

Als "V0" het adres is van de eerste geheugenlokatie van een variabele, dan kunnen we door het toepassen van de volgende regels type- en letter-informatie van elkaar scheiden:

```
T0=INT(PEEK(V0)/32)
C$=CHR$(PEEK(V0)-T0*32+32)
```

waarin "T0" het type aangeeft en "C$" de letter.

Het gebruiken van de eerste lokatie voor het opslaan van deze gegevens is zeer verstandig, want door het onderzoeken van de eerste geheugenlokatie weten we wat voor soort variabele op een bepaalde plaats van het geheugen is opgeslagen en weten we bijgevolg ook hoe we er verder mee moeten omgaan.

Dit principe wordt in onderstaande "utility" toegepast:

```
9300 LET V0=PEEK 16400+256*PEEK 16401
9310 PRINT "VARIABELE";TAB(10);"TYPE";
  TAB(20);"WAARDE"
9320 DIM C$(10)
```

```
9330 IF PEEK(V0)=128 THEN STOP
9340 LET T0=INT(PEEK(V0)/32)
9350 LET I0=1
9360 LET C$=""
9370 LET C$(10)=CHR$(PEEK(V0)-T0*32+32)

9380 IF T0=3 THEN GOTO 9450
9390 IF T0<>5 THEN GOTO 9480
9400 LET V0=V0+1
9410 LET T0=INT(PEEK(V0)/64)
9420 LET I0=I0+1
9430 LET C$(I0)=CHR$(PEEK(V0)-T0*64)
9440 IF T0=0 THEN GOTO 9400
9450 PRINT C$;TAB(20);VAL C$
9460 LET V0=V0+6
9470 GOTO 9330

9480 IF T0<>7 THEN GOTO 9520
9490 PRINT C$;TAB(10);"INDEX";TAB(20);VAL C$
9500 LET V0=V0+18
9510 GOTO 9330

9520 IF T0<>2 THEN GOTO 9640
9530 LET I0=I0+1
9540 LET T0=PEEK(V0+1)+256*PEEK(V0+2)
9450 LET C$(2)="$"
9560 PRINT C$;TAB(10);"LEN";T0;TAB(20);
9570 LET V0=V0+3
9580 IF T0=0 THEN PRINT
9590 IF T0=0 THEN GOTO 9330

9600 PRINT CHR$(PEEK(V0);
9610 LET V0=V0+1
9620 LET T0=T0-1
9630 GOTO 9580

9640 IF T0=6 THEN LET C$(2)="$"
9650 LET I0=0
9660 PRINT C$;TAB(10);"DIM(";
9670 PRINT PEEK(V0+4+I0*2)+256*PEEK(V0+5+I0*2);
9680 LET I0=I0+1
9690 IF I0<>PEEK(V0+3) THEN PRINT",";
9700 IF I0<>PEEK(V0+3) THEN GOTO 9670
9710 LET V0=V0+3+PEEK(V0+1)+256*PEEK(V0+2)
9720 PRINT ")"
9730 GOTO 9330
```

Met regel 9300 wordt "V0" gelijk gemaakt aan het adres van de startbyte van het variabelengebied van het geheugen. Door middel van de regels 9340 en 9370 wordt de variabele opgesplitst in "T0" (type) en "C$ (1)" (eerste letter van de naam). Wat er vervolgens gebeurt, is afhankelijk van het soort variabele, dus van de cijferinhoud van "T0".

Als T0 gelijk is aan drie, hebben we te maken met een eenvoudige variabele. Regel 9380 kijkt of dit het geval is en is bet antwoord bevestigend, dan stuurt een "GOTO" de computer naar regel 9450. De naam en de waarde van de variabele worden geprint.
Het printen van de naam is simpel, want deze staat immers in "C$". Het uitschrijven van de waarde kan alleen maar door gebruik te maken van de functie "VAL". Deze typische BASIC-functie is zo nuttig dat het de moeite waard is ze te kunnen toepassen. Met een "VAL"-instruktie kunnen we immers een bepaalde wiskundige uitdrukking in een "stringvariabele" uitwerken. Als we er aan denken dat de naam van een enkele variabele een speciale vorm van een wiskundige uitdrukking is, ligt het voor de hand dat de notatie "PRINT VAL C$" niets anders zal doen dan de inhoud van de variabele die in "C$" zit uit te printen. En dat is nu net wat we wilden!
Nadat deze informatie op het scherm is gezet, moeten we er voor zorgen dat de inhoud van "V0" zo wordt aangepast, dat er nu het adres van de eerste lokatie van de volgende variabele in komt te staan. Als we ons even voor de geest halen dat een eenvoudige variabele vijf geheugen-plaatsen inneemt (zie ook de ZX81 handleiding, pagina 172), dan zal het duidelijk zijn dat regel 9460 "V0" gelijk maakt aan het adres van de eerste lokatie van de eerstvolgende variabele.

Als "T0" gelijk is aan vijf, dan volgt uit de gegegeven tabel, dat we te maken hebben met een numerieke variabele met een naam, die langer is dan één letter.
Nu we het probleem van het printen van de inhoud van een numerieke variabele, waarvan we de naam kennen, hebben opgelost, worden we als enig extra probleem geconfronteerd met het opbouwen van de volledige naam uit de grootheid "C$". Daarvoor schakelen we de regels 9400 tot en met 9440 in.
Ieder aanvullend symbool van de naam van de variabele wordt door regel 9430 afgezonderd,

samen met een nieuwe waarde van de typecode "T0" (regel 9410). De volledige naam van de variabele wordt opgebouwd door ieder nieuw symbool toe te voegen aan "C$". Het laatste symbool van de naam wordt gedetecteerd door het feit dat "T0" niet gelijk is aan nul.
Nadat het laatste symbool is afgezonderd, kunnen we de gegevens printen en de waarde van "V0" op dezelfde wijze aanpassen, als bij een eenvoudige variahele is heschreven.

Als "T0" gelijk is aan zeven, hebben we te maken met een index-variabele. Het enige verschil tussen zo'n grootheid en een eenvoudige variabele is dat een index liefst 18 geheugenlokaties nodig heeft voor het opbergen van zijn momentele waarde. U moet, na de vorige uitleg, in staat zijn te beredeneren hoe we door middel van de instrukties in de regels 9490 en 9500 dit soort variabele behandelen.

Als "T0" gelijk is aan twee, krijgen we een "string" voorgeschoteld. Omdat de naam van een "string" per definitie slechts één letter lang kan zijn, hoeven we alleen een "$"-symbool aan de inhoud van "C$" toe te voegen, hetgeen gebeurt in regel 9550.
Helaas kunnen we geen beroep doen op de "VAL"-functie voor het printen van de momentele waarde van de variabele. Deze funktie kan immers alleen wiskundige uitdrukkingen aan!
De lengte van de "string" wordt in de twee geheugenlokaties opgeslagen, die volgen op de naam.

geheugenlokatie

1	2 3	4 5	... lengte + 3
naam	lengte	string	

Als we deze twee plaatsen gaan "PEEK"-en (regel 9450), kunnen we de lengte van de "string" printen door middel van regel 9560. Met deze informatie kunnen we nadien ieder symbool van de "string" "PEEK"-en en door de "CHR$"-functie alles op het scherm schrijven. Als het aantal symbolen gelijk wordt aan de lengte van de "string" weten we dat we klaar zijn (regels 9580 en 9590).

Een "T0"-waarde van vier komt overeen met een nummer-array, een "T0"-waarde van zes duidt aan dat we te maken hebben met een symbolen-array. Het enige verschil in behandeling is dat een "$"-symbool aan de enkele letter wordt toegevoegd in het geval van een "string-array" (regel 9640).
Het heeft geen enkele zin de momentele waarde van dit soort variabelen op het scherm te zetten. Het resultaat zou een overvloed aan informatie zijn, waardoor het gewenste overzicht verloren zou gaan. Zinvoller lijkt het in dit geval de dimensies van de "string" op het scherm te zetten.
Dan moeten we wel eerst bekijken hoe de ZX81 "array's" in zijn geheugen opslaat.
Uit hoofdstuk 27 van de handleiding van de machine volgt dat de vierde byte het nummer van de "array" bewaart.
De afmeting (dimensie) van een array wordt opgeslagen in twee bytes.

Bytes

1	2 3	4	5 6	7 8	...
naam	lengte	nr. van array	1ste dim.	2de dim.	

Door het "PEEK"-en van de inhoud van de vierde byte kunnen we vaststellen of de afmetingen van alle array's zijn uitgeprint, regel 9690 en 9700. De afmeting van het array volgt uit regel 9670 en het aantal wordt in de grootheid "I0" opgeslagen door regel 9680.
Nadat we al deze informatie op het scherm hebben gezet, wordt de inhoud van "V0" aangepast. De adreswaarde van de eerstvolgende variahele verschijnt in "V0" door bij de oorspronkelijke waarde de inhoud van de tweede en derde byte op te tellen. Deze plaatsen bevatten immers de totale lengte van het array.

Er blijven nu slechts twee feiten te verklaren.
Op de eerste plaats moet het programma op de een of andere manier te weten komen, of alle variabelen aan de beurt zijn geweest. Dat is geen punt, de laatste geheugenplaats uit het variabelengebied bevat immers de waarde 128 en dit kunnen we dus zeer eenvoudig kontroleren door regel 9330.
Op de tweede plaats zouden we ons kunnen afvragen, waarom we voor "C$" een "DIM" kiezen en geen "string", terwijl dit laatste toch veel eenvoudiger zou zijn. Welnu, een "string" kan de hoeveelheid voor zichzelf noodzakelijke geheugenruimte wijzigen gedurende het uitwerken van een programma. Zouden we in dit

hulpprogramma een "string" gebruiken, dan zou het variahelengebied van het geheugen telkens als de "string" van lengte zou veranderen opnieuw worden gerangschikt, hetgeen natuurlijk desastreus is voor een programma waarmee we juist de inhoud van dit geheugendeel willen analyseren!
Waarschijnlijk vindt u deze "utility" tamelijk moeilijk om te begrijpen. Analyse van ieder zelfstandig onderdeel zal hopelijk wat licht in de duisternis werpen.
Als dat bij u nog niet het geval is, mag dat geen reden zijn om geen gebruik te maken van dit zeer nuttige programma.
Hier volgt een deel van een programma en de manier waarop deze "utility" de gebruikte variabelen in tabelvorm verwerkt.

```
1 LET A=100
2 LET MAX=00120
3 LET A$="MIKE"
4 FOR I=1 TO 5
5 DIM Z(1,7,9)
          •
          •
          •
```

VARIABELE	TYPE	WAARDE
A		100
MAX		120
A$	LEN=4	MIKE
I	INDEX	1
Z	DIM(1,7,9)	
	•	
	•	
	•	

"Utility" nummer 3: het hernummeren
De derde en laatste "utility" die we gaan bespreken is ook weer een vrij kort programma, waarmee we de nummering van de regels uit een programma kunnen aanpassen.
In de praktijk komt het bijna nooit voor dat we een lang programma in één keer zonder systeemfouten in de computer tikken.

Er gaat altijd wel iets mis en dan moeten we extra regels tussenvoegen, waardoor de nummering van de regels niet meer klopt. Het zou dus erg handig zijn, als we konden beschikken over een programma dat automatisch alle regels van het foutloze programma kan hernummeren.
Dat is echter niet zo eenvoudig in BASIC te programmeren. Niet alleen moeten alle regels nieuwe nummers krijgen (dat zou nog wel te doen zijn) maar bovendien moet iedere "GOTO"- of "GOSUB"-instruktie worden aangepast aan de nieuwe nummering. Zou men dat niet doen, dan zouden de oude regelnummers in de "GOTO"- en "GOSUB"-bevelen naar volkomen foute instrukties verwijzen en het programma zou niet werken.
Natuurlijk is ook dat te programmeren, maar de "utility" wordt dan zo lang, dat het nauwelijks nog de naam "hulpprogramma" verdient!
Vandaar het volgende compromis: we maken een hulpprogramma, dat alle regels van een nieuwe logische nummering voorziet, maar de regelnummers achter de "GOTO"- en "GOSUB"-instrukties ongewijzigd laat. Het programma stelt dan een tabel op, waarin de oude en nieuwe regelnummers met elkaar worden vergeleken.
Aan de hand van deze tabel en met behulp van de "EDIT"-instruktie kunnen we vrij snel alle "GOTO"- en "GOSUB"-commando's aanpassen.
Alvorens we de werking van het programma kunnen begrijpen, is het noodzakelijk even wat informatie te geven over de manier waarop de machine BASIC-instrukties in het programmagedeelte van het geheugen opslaat.
Uit hoofdstuk 27 van de handleiding kunnen we de volgende gegevens afleiden:

2 bytes	2 bytes	tekst	nieuwe-regel
regelnr.	tekstlengte + nieuwe regel		

Hieruit blijkt, dat de twee eerste geheugenlokaties de gegevens over het nummeren van de regels bevatten. Opmerkenswaard is wel, dat het regelnummer met het meest beduidende bit (cijfer) in het eerste byte wordt opgeslagen, dus tegengesteld aan alles wat we tot nu toe over het opslaan van getallen groter dan 256 hebben geleerd!
Vandaar dat we onderstaande "PEEK"-instruktie moeten gebruiken:

```
256*PEEK V0 + PEEK (V0+1)
```

waarbij "V0" het adres van de eerste geheugenlokatie bevat. Als we het regelnummer een andere waarde willen geven, moeten we de twee

lokaties "POKE"-en met de nieuwe gegevens. We zullen in de eerste plaats rekening moeten houden met de beperkte opslag-capaciteit van een byte en getallen groter dan 256 moeten opsplitsen in twee delen:

hoogstwaardige byte = INT(S0/256)

en

laagstwaardige byte = S0-256*INT(S0/256)

waarbij "S0" staat voor het nieuwe regelnummer.
Na het "POKE"-en van het nieuwe regelnummer moeten we naar de volgende regel verhuizen. Dat kan, door de derde en vierde geheugenlokatie te "PEEK"-en (deze plaatsen bevatten de informatie over de lengte van de regel) en deze gegevens op te tellen bij de originele "V0"-waarde.
We zijn nu rijp om geconfronteerd te worden met het programma van "utility" nummer 3!

```
9800 LET V0=16509
9810 PRINT "START=";
9820 INPUT S0
9830 PRINT S0
9840 PRINT "STAP=";
9850 INPUT I0
9860 PRINT I0
9870 PRINT
9880 PRINT "OUD";TAB(6);"NIEUW"
9890 LET L0=256*PEEK V0 + PEEK (V0+1)
9900 IF L0>=9000 THEN STOP
9910 PRINT L0;TAB(6);S0
9920 POKE V0,INT(S0/256)
9930 POKE V0+1,S0-256*INT(S0/256)
9940 LET S0=S0+I0
9950 LET V0=V0+4+PEEK(V0+2)+256*PEEK(V0
  +3)
9960 GOTO 9890
```

Met regel 9800 maken we "V0" gelijk aan het adres van de eerste byte van het programmagebied uit het geheugen.
Nadien stellen we het getal vast, waarmee het hernummeren moet beginnen ("S0") en ook de stap-grootte tussen iedere regel ("I0"). Een en ander gebeurt aan de hand van de regels 9810 tot en met 9870. Regel 9890 berekent het momentele regelgetal "L0".

Deze grootheid wordt samen met het nieuwe regelnummer uitgeschreven onder bevel van regel 9910.
Nadien wordt het nieuwe regelnummer door middel van twee "POKE"-instrukties in de juiste bytes ingeschreven (regels 9920 en 9930). De grootheid "S0" wordt vermeerderd met de gekozen stap-grootte "I0" (regel 9940) en met de volgende regel maakt men "V0" gelijk aan het adres van de eerste byte van de volgende instruktieregel.
Het programma stopt als "L0" gelijk wordt aan 9000. Het heeft immers geen enkele zin de regelnummers van deze of een van de twee andere "utility's" te gaan hernummeren!
Alvorens u deze "utility" op gelijk welk programma kunt loslaten, moet u wel twee gegevens intoetsen. Het nieuwe nummer van de eerste regel (meestal 10) en de gewenste stapgrootte tussen de regels (meestal ook 10). Het eerste gegeven wordt ingetoetst, nadat de mededeling "START" op het scherm verschijnt, de tweede informatie wordt verwacht na de boodschap „STEP".
Een voorbeeld van het resultaat van deze "utility":

```
START=10
STAP=10
OUD   NIEUW
10    10
11    20
12    30
13    40
14    50
```

Naast het feit dat dit eenvoudige hernummeringsprogramma erg nuttig is, kan het ook de basis vormen voor een meer geavanceerd hernummeringsprogramma. U kunt het bijvoorbeeld zo aanpassen, dat alleen maar een bepaald gedeelte van het programma onder handen wordt genomen, of u kunt proberen het moeilijke probleem van de "GOSUB"- en "GOTO"-instrukties ermee te vereenvoudigen!

4. GRAFISCHE MOGELIJKHEDEN MET 16K

Het ter beschikking hebben van veel meer geheugenruimte heeft grote gevolgen op de manier waarop men grafische voorstellingen op de ZX81 kan programmeren.
Een van de meest voor de hand liggende voordelen is dat men het volledige scherm kan inschakelen, zonder dat men het gevaar loopt dat het geheugen overloopt. Een volledig gebruikt scherm kost ongeveer 700 geheugenplaatsen. Het is duidelijk dat er in de 1K-uitvoering dan niet zo erg veel geheugenruimte meer overblijft! Daarnaast zal blijken dat de 16K-machine op een geheel andere manier omgaat met de beeldmogelijkheden dan de 1K-uitvoering en dat we dit feit kunnen gebruiken om door middel van vrij geavanceerde programmeertechnieken vrij ingewikkelde grafische voorstellingen te kunnen tekenen.

Een duikbootspelletje
Dit spel is tamelijk eenvoudig. Er worden twee schepen op het scherm getekend, een oppervlakteschip en een onderzeeër. Het oppervlakteschip kan dieptebommen lanceren en het is de bedoeling dat we met zo weinig mogelijk bommen proberen de onderzeeër te vernietigen.
De ideeën achter het programma zijn recht-toe-recht-aan.
De twee schepen worden op het scherm geprint door eerst op een bepaalde plaats een vorm te tekenen, nadien deze vorm uit te wissen om het symbool op een iets afwijkende plaats opnieuw te tekenen.
Omdat de schepen zich uiteraard horizontaal en zelfs rechtlijnig voortbewegen, kan het programma worden vereenvoudigd door de volgende truuk.

```
10 FOR X=0 TO 30
20 PRINT AT 3,X;" *";
30 NEXT X
```

We zien een sterretje dat van links naar rechts over de bovenste helft van het scherm "loopt". Door het symbool op te bouwen uit de combinatie van een spatie en een sterretje en de positie van dit symbool op het scherm te variëren door het laten stijgen van de waarde van de grootheid "X", ontstaat een bewegend beeld. Bij iedere nieuwe "PRINT" wordt de oude positie van het sterretje immers ingenomen door de spatie!
Deze techniek kunnen we bijna altijd gebruiken. Het enige waarop we moeten letten is dat er zich rondom het symbool voldoende spaties bevinden om ervan verzekerd te zijn dat het oude symbool volledig door spaties wordt vervangen bij de nieuwe "PRINT"-instruktie. Moeilijker wordt het als het objekt zich niet rechtlijnig voortbeweegt. In het later in dit hoofdstuk te behandelen "squash"-programma maken we ook van deze truuk gebruik.

Er moeten nu nog twee problemen worden opgelost: hoe kunnen we vaststellen of de duikboot is geraakt en hoe kunnen we in zo'n geval een soort "explosie" laten plaatsvinden, waarna de duikboot van het scherm verdwijnt? Het volledige programma luidt als volgt.

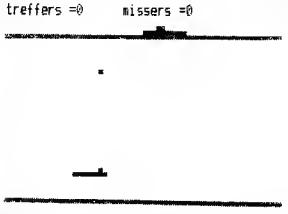
```
 10 LET MC=0
 20 LET HC=0
 30 LET F=0
 40 LET H=0
 50 LET XS=0
 60 GOSUB 600
 70 LET X=3
 80 LET YS=INT(RND*5)+15
 90 LET Y=2
100 GOSUB 200
110 PRINT AT 0,0;"TREFFERS=";HC;TAB(10);"MISSERS
  =";MC
120 IF F=0 THEN LET XD=2*X
130 IF F=0 THEN LET YD=35
140 IF INKEY$="F" AND F=0 THEN LET F=1

150 IF F=1 THEN GOSUB 300
160 GOSUB 400
170 IF H=0 THEN GOTO 100
180 GOSUB 500
190 GOTO 30
```

```
200 PRINT AT Y,X;" [6][ ][6][6]";
210 LET X=X+1
220 IF X>25 THEN LET X=0
230 IF X=0 THEN PRINT AT Y,25;"      ";
240 RETURN

300 UNPLOT XD,YD
310 LET YD=YD-1
320 PLOT XD,YD
330 IF YD=2 THEN LET MC=MC+1
340 IF YD=2 THEN LET F=0
350 IF F=0 THEN UNPLOT XD,YD
360 IF ABS(XD-2*XS-6)>6 THEN RETURN
370 IF YD<>43-2*YS THEN RETURN
380 LET H=I
390 RETURN

400 PRINT AT YS,XS;" [6][6][W]";
410 LET XS=XS+RND
420 IF XS>27 THEN LET XS=0
430 IF XS=0 THEN PRINT AT YS,27;"      ";
440 RETURN

500 LET HC=HC+1
510 FOR I=1 TO 20
520 PRINT AT YS,XS;"[A][A][A][A]";
530 PRINT AT YS,XS;"      ";
540 NEXT I
550 RETURN

600 CLS
610 FOR I=0 TO 31

620 PRINT AT 3,I;"[S]";
630 PRINT AT 21,I;"[D]";
640 NEXT I
650 RETURN
```

Daar waar grafische symbolen ingetoetst moeten worden, is dit in het programma aangegeven met vierkante haakjes rond de te bedienen toetsen. Zo bedoelen we met [A] het grafische symbool van de A-toets en met [ ] een omgekeerde spatie, dus een zwart vierkantje. Als u dus in dit en in de hieropvolgende programma's een symbool tussen vierkante haakjes aantreft, dan moet u de computer in zijn "grafiekstand" zetten. Niet vergeten eerst de "shift"-toets in te drukken!
In de regels 230, 430 en 530 moet u tussen de aanhalingstekens telkens 4 spaties intoetsen.

Hoe werkt dit programma?
Het programma start met het op nul zetten van alle variabelen en het tekenen van de zeebodem en zeespiegel, met behulp van subroutine-600. De diepte, waarop de onderzeeër gaat varen, wordt door middel van een willekeurig getal in regel 80 vastgelegd. Nadien wordt het oppervlakteschip door subroutine 200 getekend op positie "X, Y" en de onderzeeër door subroutine 400 op de positie "XS, YS". Beide schepen varen in dezelfde richting. De snelheid van de duikboot wordt door regel 410 op een willekeurige waarde vastgelegd. De dieptebom wordt gelanceerd op het moment dat de "F"-toets wordt ingedrukt. Het al dan niet bedienen van deze toets wordt in regel 140 gecontroleerd door de "INKEY$"-functie. De variabele "F" wordt 1 en de positie van de bom wordt in "XD, YD" opgeslagen.
Door middel van subroutine 300 houdt het programma de positie van de bom in de gaten en zorgt er bovendien voor dat het projektiel blijft bewegen.
We moeten wel bedenken dat de bom op het scherm wordt gezet door middel van "PLOT/UNPLOT"-instrukties en de schepen met "PRINT AT"-bevelen worden getekend. De coördinatensystemen kloppen dus niet! Zoals we weten, verdeelt "PLOT" het scherm in 64 bij 44 posities en werkt "PRINT AT" met 32 bij 22 symboolplaatsen. Het zal duidelijk zijn, dat "XD" en "YD" het dubbele zijn van een gelijkwaardige waarde van "X" en "Y". Ook moeten we rekening houden met het feit dat de waarde van "YD" wordt gemeten vanaf de onderkant van het scherm en de waarde van "Y" vanaf de bovenkant. Al met al is het dus niet zo eenvoudig om een systeem te verzinnen dat de coördinaten van duikboot en dieptebom met elkaar kan vergelijken.
Met behulp van de regels 120 en 130 worden de variabelen "XD" en "YD" gelijkgemaakt aan de coördinaten van het oppervlakteschip, zodat de bom vanaf de momentele plaats van het schip wordt gelanceerd. Met de regels 360 en 370 worden de coördinaten van de onderzeeër en van de bom met elkaar vergeleken. Het vergelijken van de X-coördinaten volgt na een vermenigvuldiging met twee. Voor het vergelijken van de Y-coördinaten moet men echter "2*YS" van 43 aftrekken. Over de overige programmastappen is niet zo veel op te merken, het enige

waar u in de toekomst misschien nog eens gebruik van kunt maken is de manier waarop de duikboot in de regels 510 tot en met 540 wordt "vernietigd".
In de figuur is te zien hoe de symbolen van dit spel op het scherm verschijnen.



Squash
Als tweede voorbeeld van een spel waarbij we, dankzij de 16K geheugenkapaciteit, het hele scherm kunnen gebruiken, geven we een volledig uitgewerkt programma voor het spelen van het zogenoemde "squash"-balspel.

```
 10 LET BALL=0
 20 LET BALL=BALL+1
 30 CLS
 40 LET A=10
 50 LET B=10
 60 LET V=1
 70 LET W=1
 80 LET X=10
 90 LET Y=15
100 GOSUB 500
110 PRINT BALL
120 GOSUB 200
130 GOSUB 700
140 GOSUB.300
150 IF B<>21 THEN GOTO 120
160 GOTO 20

200 LET A$=INKEY$
210 IF A$="S" THEN LET X=X-1
220 IF A$="B" THEN LET X=X+1
230 RETURN

300 PRINT AT B,A;" ";
310 LET A=A+V
320 LET B=B+W
330 IF A=31 OR A=0 THEN LET V=-V
340 IF B=1 THEN LET W=-W
350 IF B+1=Y THEN GOSUB 400
360 PRINT AT B,A;"[]";
370 RETURN

400 LET R=A-X
410 IF R<1 OR R>3 THEN RETURN
420 LET W=-W
430 RETURN

500 FOR I=0 TO 31
510 PRINT AT 0,I;"[]";
520 NEXT I
530 RETURN

700 PRINT AT Y,X;"[][][]";
710 RETURN
```

In feite passen we hier dezelfde technieken toe, als voor de 1K-uitvoering gelden.
Het programma tekent allereerst de muur van het speelterrein op het scherm, met subroutine 500.
Nadien tekent subroutine 300 de bal op positie "A, B" en berekent telkens opnieuw de coördinaten van dit punt aan de hand van de horizontale snelheid "V" en de vertikale snelheid "W". Als de bal tegen een van de "muren" botst, dus van het scherm zou vliegen, zorgt deze subroutine voor het inverteren van de vertikale of horizontale snelheid, zodat de bal terugkaatst.
Het slaghout wordt bestuurd door subroutine 700. De positie wordt vastgelegd door de variabelen "X, Y". Het is niet noodzakelijk de vorige positie van de "bat" te wissen, daar er aan weerszijden van het symbool spaties zijn opgenomen, die daar op de bekende manier automatisch voor zorgen.
Het enige nieuwe in dit programma is de manier waarop het slaghout wordt bestuurd en de wijze waarop men de bal op het hout laat kaatsen. Het zal duidelijk zijn dat de "bat" heen en weer kan worden gestuurd, door het veranderen van de horizontale coördinaat, afhankelijk van de toets die men indrukt. Als de "linker pijl"-toets wordt bediend, dan wordt de waarde van "X" met een eenheid verlaagd, waardoor het slaghout een schermplaats naar links gaat.

Drukt men daarentegen op de "rechter pijl"-toets, dan wordt de waarde van "X" met een eenheid vermeerderd. hetgeen tot gevolg heeft dat het symbool op het scherm naar rechts gaat. Deze procedure wordt uitgevoerd door subroutine 200.

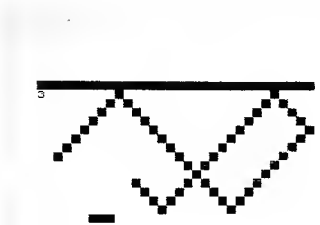
Het valt op, dat er in deze suhroutine geen maatregelen genomen zijn, die er voor zorgen dat de "bat" niet van het scherm kan "vallen". Dit heeft te maken met de traagheid van de ZX81. Het invoeren van zo'n testprocedure zou de uitvoering van dit programma nog meer vertragen.

Subroutine 400 houdt zich bezig met het terugkaatsen van de bal op het slaghout. Als de bal dezelfde vertikale positie als het slaghout heeft, dan roept regel 350 genoemde subroutine te hulp om te kontroleren of ook de horizontale positie overeenkomt. Is dat het geval, dan "kaatst" de bal terug, door het inverteren van zijn vertikale snelheid.

Als u de bal mist, detecteert regel 150 het overschrijden van de onderste grens van het speelveld.

Na controle wordt de computer naar regel 20 gestuurd, waar een nieuw spel start. Het spel wordt gespeeld door de instrukties in de regels 120, 130 en 140, die steeds opnieuw door de "GOTO 120" in regel 150 worden doorlopen. Regel 120 roept de suhroutine op, die de "bat" dirigeert. Regel 130 zorgt door middel van subroutine 700 voor het uittekenen van het slaghout en regel 140 neemt de bal en het "kaatsen" vooJ zijn rekening.

Onderstaande afbeelding geeft een indruk van het door dit programma getekende beeld, waarhij valt op te merken dat de regels die ervoor zorgen dat iedere vorige positie van de bal wordt gewist, waren uitgeschakeld.



Met de in dit hoofdstuk behandelde technieken kan men eigen spelletjes gaan programmeren en zelfs de voorgestelde programma's (duikboot en squash) verbeteren en uitbreiden. Toch moeten we rekening houden met het feit, ook al worden we niet meer in onze mogelijkheden geremd door gebrek aan geheugen (de 1K-uitvoering) we wél worden geconfronteerd met de traagheid van de 16K-machine!

Schermindeling

Als we de ZX81 inschakelen, zal de computer gaan onderzoeken hoeveel geheugen er ter beschikking staat.

Als er minder dan 3.5K voorradig is, zal de machine een zo zuinig mogelijke schermindeling, een zogenoemde "minimum screen" gaan opbouwen. Een "minimum screen" bestaat in eerste instantie uit 25 symboolplaatsen, die "newline" worden genoemd. De eerste symboolplaats hiervan geeft het begin van het schermgedeelte van het geheugen aan. De 24 volgende symbolen kunnen het einde van een schermregel aanduiden. Als men symbool-informatie op het scherm zet, worden deze symholen geplaatst tussen de bijbehorende "newline"-symbolen. Op deze manier weet de computer de juiste schermbeeldplaats voor ieder ingetikt symbool. Op iedere regel kunnen 32 symbolen worden geschreven. Als er tussen twee "newline"-symbolen minder dan 32 symbolen staan, dan vult de machine dit aantal automatisch met spaties aan tot 32.

Hoewel een spatie geen zichthare gevolgen heeft, is het voor de computer natuurlijk een volwaardig symbool en neemt het net zo veel plaats in als bijvoorbeeld de letter A.

Er zijn dus twee systemen, waarmee de instructie voor een volledig lege regel in het geheugen kan worden bewaard; ofwel door 32 spatiesymbolen tussen twee "newline's" op te slaan, ofwel door twee "newline's" achter elkaar te plaatsen. In het eerste geval stuurt de computer 32 spaties naar het scherm, want die staan in het geheugen! In het tweede geval berekent de machine het aantal spaties, nodig voor een volle regel en stuurt dat aantal naar het scherm.

Het zal duidelijk zijn, dat we volgens het eerste systeem 32 geheugenplaatsen extra nodig heb-

ben voor hetzelfde resultaat! Stel, dat het begin van het beeldgeheugen er als volgt uit ziet.

-NL-NL-H-I-SP-T-H-E-R-E-NL-NL-. . .-NL-

waarhij "NL" staat voor een "newline"-code, "SP" voor een spatie en de overige letters vervangen mocten worden door hun codenummers (zeg: 45 voor H). De computer interpreteert deze geheugeninhoud als volgt. De eerste "NL" dient er alleen voor dat de computer weet dat het heeldgedeelte van het geheugen nu begint. De tweede "NL" markeert het einde van de eerste regel. De computer stelt vast dat er tussen beide "NL"-codes geen symbolen staan en bijgevolg worden 32 spaties naar het scherm gestuurd voor de eerste regel.
Na de tweede "NL" volgen acht symboolcodes. Deze worden op de tweede regel afgedrukt, gevolgd door 24 spaties. De tweede regel is ook gevuld. Dit gaat zo door, tot het gehele scherm is volgeschreven.
Deze beeldophouw is tamelijk ingewikkeld, maar heeft wel als groot voordeel dat er geen geheugenruimte verloren gaat voor het opslaan van onnodige spaties.

Als er echter meer dan 3,5K aan geheugen ter beschikking staat, gebruikt de machine een veel eenvoudiger systeem.
Alle 24 regels worden dan volledig met spaties gevuld en dat kost uiteraard veel meer geheugen!
Wel worden de "newline"-symholen nog steeds gebruikt voor het afbakenen van het beeldgedeelte van het geheugen en voor het aangeven van het begin van een nieuwe regel. De basisstruktuur is bij de 16K-uitvoering dus gelijk aan die van de 1K-versie.
Als men informatie op het scherm schrijft, worden de bijbehorende spatiecodes in het geheugen vervangen door codes van de ingetoetste symbolen. De totale gebruikte geheugenomvang blijft dus gelijk. Toch is er op deze regel één uitzondering! Als we een "SCROLL"-instruktie geven, schuift het volledige beeld een regel op naar boven en wordt een nieuwe regel aan het beeldgeheugen toegevoegd, bestaande uit één "newline"-symbool. De eerste regel verdwijnt dan uit het beeldgeheugen. Iedere "SCROLL"-instruktie introduceert dus een korte geheugenregel op het scherm. Na 22 "SCROLL's" is het gehele beeldgeheugen op dezelfde wijze opgehouwd als bij de 1K-uitvoering. Alle overbodige spatiesymbolen zijn dan verwijderd.
Na een reset door middel van de "CLS"-functie, gaat het geheel terug naar de oorspronkelijke situatie.

Het "PEEK"- en "POKE"-en van het beeld
Uit de vorige paragraaf weten we, dat de 16K versie een bepaald vast geheugendeel reserveert voor het heeld, als we geen "SCROLL" gebruiken.
De eerste lokatie van het beeldgedeelte van het geheugen ligt in de "D-FILE" en daar bevindt zich een "newline"-symbool. De volgende 32 lokaties worden gebruikt voor het opslaan van de codes van alle symbolen van de eerste regel. door u ingetoetst, of de originele spaties van de basisindeling.
De eerstvolgende lokatie bevat weer een "newline" en deze indeling gaat zo verder tot en met het einde van de laatste regel van het beeldgedeelte.
Het gevolg van deze indeling is dat we zeer eenvoudig een verband kunnen leggen tussen een bepaalde plaats op het scherm en de geheugenlokatie, waarin de code zit van wat er op deze plaats is weergegeven.
En als we eenmaal het adres van zo'n lokatie te pakken hebben. kunnen we door "PEEK" en "POKE" te weten komen wat er op die bepaalde plaats op het scherm staat of de inhoud van deze schermplek wijzigen.
Als "D" het adres is van de startlokatie van het beeldgedeelte, dan is het adres van de geheugenlokatie die overeenkomt met kolomnummer "X" en regelnummer "Y":

```
D+X+Y*33+1
```

Iedere regel is immers 32 symbolen plus een „newline" lang en het geheugendeel start met een „newline".
Met deze formule kunnen we op iedere gewenste plaats van het scherm (ook op de normaliter niet te gebruiken twee onderste regels) nieuwe symbolen "POKE"-en.
Proheer maar eens onderstaand programma:

```
10 CLS
20 LET D=PEEK 16396+256*PEEK 16397
```

```
30 FOR X=0 TO 31
40 FOR Y=0 TO 23
50 LET A=D+X+Y*33+1
60 POKE A,128
70 NEXT Y
80 NEXT X
```

Het volledige scherm wordt volgeschreven met vierkantjes, ook de "verboden" twee onderste regels!
Algemeen gesteld, kan men het bekende bevel:

```
PRINT AT Y,X;A$;
```

waarin "A$" staat voor een enkel symbool, vervangen door:

```
POKE D+X+Y*33+1,CODE(A$)
```

waarin "D" staat voor het adres van de eerste byte uit de beeldzone van het geheugen.
Misschien vraagt u zich af, waarom we op zo'n ingewikkelde manier iets op het scherm zouden willen schrijven. Er zijn een aantal goede redenen te verzinnen, waaronder de mogelijkheid toegang te krijgen tot de onderste twee regels en de noodzaak sneller te kunnen printen.
Nochthans is de mogelijkheid het scherm te kunnen "PEEK"-en veel nuttiger, want het is de enige manier om uit te zoeken wat er op een bepaalde plek van het scherm geschreven staat. Om de code terug te vinden van het symbool op geheugenlokatie "X, Y" staat de volgende formule ter beschikking.

```
LET C=PEEK(D+X+Y*33+1)
```

waarin "D" staat voor het adres van de eerste lokatie in het beeldgeheugen en "C" voor de code van het op te zoeken symbool.
In de volgende paragraaf bespreken we een spel, waaruit duidelijk het voordeel blijkt van het door middel van een "PEEK" opzoeken van wat er op een bepaalde plaats van het scherm aanwezig is.

Een doolhofspel

Onderstaand programma geeft de ZX81 versie van een zeer populair spel, zo populair zelfs dat we het bij zowat alle micro-computers terugvinden.
Door middel van de vier "pijltjes"-toetsen kunt u een sterretje over het scherm laten bewegen. Als het spel start, bevindt dit sterretje zich in de onderste rechter hoek van het beeld. De kunst is dit symbool naar de linker bovenhoek te sturen, hetgeen niet zo gemakkelijk is als het lijkt, want op het scherm wordt een soort doolhof opgebouwd, bestaande uit een steeds wisselend patroon van donkere vlekjes! De kunst is dan uiteraard het sterretje zo snel mogelijk tussen de openingen van de doolhof te sturen en een botsing met een vierkantje te vermijden. Met de kennis die we tot nu toe hebben vergaard, moet het mogelijk zijn zélf het programma op te stellen. Aan de slag, dus! Het enige probleem is te bepalen wanneer het sterretje door een vierkant de weg wordt afgesneden en hierbij komt onze kennis over het "PEEK"-en van de beeldinhoud goed van pas!

Nog een tip: het is niet noodzakelijk de positie van alle vierkantjes op het scherm op te slaan.
Als u klaar bent, kunt u uw werkstuk vergelijken met onderstaand programma.

```
 10 CLS
 20 PRINT "MOEILIJKHEIDSGRAAD 1-9 ?"
 30 INPUT L
 40 LET L1=L/10
 50 CLS
 60 GOSUB 100
 70 GOSUB 300
 80 GOSUB 500

100 LET D=PEEK 16396+256*PEEK 16397
110 RETURN

200 LET C=PEEK(D+X+Y*33+1)
210 RETURN

300 FAST
310 GOSUB 1000
320 FOR I=1 TO 20*L+40
330 LET X=RND*29+1
340 LET Y=RND*19+1
350 PRINT AT Y,X;"[A]";
360 NEXT I
370 SLOW

380 PRINT AT 1,1;"$";
390 PRINT AT 20,30;"*";
400 RETURN
```

```
500 LET XC=30
510 LET YC=20
520 LET M=0
530 LET X=XC
540 LET Y=YC
550 LET A$=INKEY$
560 GOSUB 900
570 IF A$="" THEN GOTO 550
580 IF A$="5" THEN LET X=XC-1
590 IF A$="8" THEN LET X=XC+1
600 IF A$="6" THEN LET Y=YC+1
610 IF A$="7" THEN LET Y=YC-1
620 GOSUB 200
630 IF C=13 THEN GOTO 800
640 IF C<>0 THEN GOTO 530
650 PRINT AT YC,XC;" ";
660 LET XC=X
670 LET YC=Y
680 PRINT AT YC,XC;"$";
690 LET M=M+1
700 GOTO 530
900 CLS
810 PRINT AT 0,2;"U HEEFT HET GEHAALD IN ";M;
    " BEURTEN"
820 PRINT "NOG EEN SPEL J/N"
830 INPUT A$
840 IF A$(1)="J" THEN RUN
850 IF A$(1)<>"N" THEN GOTO 820
860 STOP

900 IF RND>L1 THEN RETURN
910 LET C$=" "
920 IF RND>.5 THEN LET C$="[A]"

930 PRINT AT RND*19+1,RND*29+1;C$
940 PRINT AT 1,1;"$";
950 RETURN

1000 FOR I=0 TO 31
1010 PRINT AT 0,I;"[]";AT 21,I;"[]";
1020 NEXT I
1030 FOR I=0 TO 21
1040 PRINT AT I,0;"[]";AT I,31;"[]";
1050 NEXT I
1060 RETURN
```

Bij dit programma starten we met het opvragen van de moeilijkheidsgraad "L", een factor die bepaalt hoeveel vierkanten er worden gebruikt voor het blokkeren van het sterretje. Met deze informatie bouwen we door subroutine 300 de doolhof op. Hoe dat precies in zijn werk gaat, zal wel duidelijk zijn. Eerst doen we een beroep op subroutine 1000, waarmee we het speelveld afgrenzen. Nadien tekenen we de vierkanten [A] op willekeurige plaatsen op het scherm door middel van de "PRINT AT" van regel 350. Let op het gebruik van "FAST" en "SLOW", waarmee de trage berekening van alle posities aan het oog wordt onttrokken en de doolhof zo snel mogelijk op het scherm wordt getekend. Tot slot tekent deze subroutine een "$"-symbool in de linker bovenhoek, als "doel waarna we streven" en het sterretje in de startpositie "20, 30".

Het eigenlijke spel begint met een beroep op subroutine 500. Na het vaststellen van enige beginwaarden met "LET"-instrukties, wordt door middel van regel 550 gekeken of er op een van de "pijltjes"-toetsen wordt gedrukt. De waarden van "XC" en "YC", die de momentele plaats van het sterretje bepalen, worden vervangen door "X" en "Y", de nieuwe coördinaten van het symbool, rekening houdend met de toets die is ingedrukt. Nadien moeten we gaan bepalen of deze nieuwe positie van het sterretje wel kan, met andere woorden of er op deze plaats al niet een vierkantje is getekend. Als er op deze positie een spatie staat, kan het sterretje er naar toe worden verplaatst. Als er gelijk welk ander symbool aanwezig is, moet de "zet" worden afgekeurd. Dat doen we uiteraard door het "PEEK"-en van de geheugeninhoud, door middel van subroutine 200 die door regel 620 wordt opgeroepen. De code van het op die plek aanwezige symbool wordt opgeslagen in "C". Er zijn nu een aantal mogelijkheden.

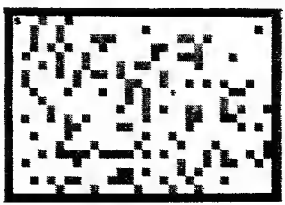
Als de code gelijk is aan 13, de code van het $-symbool, dan zijn we blijkbaar in onze opzet geslaagd en wordt het spel door regel 800 beëindigd.

Als de waarde in "C" niet gelijk is aan 0, de code voor een spatie, dan wordt dit door regel 640 gededecteerd en wordt de "zet" verworpen.

Als de code wél gelijk is aan 0, dan wordt de oude positie van het sterretje door midddel van regel 650 gewist en op de nieuwe coördinaten "XC, YC" geprint dankzij regel 680. De grootheid "M", die het aantal "zetten" registreert, wordt met 1 verhoogd en regel 700 zorgt ervoor, dat het proces zich herhaald voor de volgende "zet".

Wat we nu nog moeten bespreken, zijn subrou-

tine 900 (verantwoordelijk voor het op- en afbouwen van de wegversperringen) en de regels 800 tot en met 860 (die de resultaten van het spel op het scherm weergeven en nagaan of er behoefte aan een nieuw spel bestaat).
Beide systemen zijn gemakkelijk te begrijpen.
Het gebruik van een "PEEK" bij het bepalen of het sterretje op een vrije positie terecht komt, heeft als bijkomend voordeel, dat we geen extra "IF"-instrukties hoeven te gebruiken om het sterretje binnen het speelveld te houden. De doolhof is immers omringd door een aaneengesloten reeks vierkantjes en hierop is natuurlijk het "C"-codesysteem ook van toepassing!
Onderstaande afbeelding geeft een idee van de presentatie van dit spel op de monitor.



Met deze nieuwe ervaring moet u in staat zijn om door middel van "PEEK"- en "POKE"-instrukties de werkelijkheid nabijkomende beelden op het scherm te toveren. Een goede oefening is bijvoorbeeld het uitwerken van het squash-programma met "PEEK" en "POKE".

Het gebruik van "SCROLL" in grafiekprogramma's
We hebben in dit hoofdstuk een groot aantal technieken geïntroduceerd, waarmee we bewegende beelden kunnen opwekken. Toch zitten we met nog één moeilijk op te lossen probleem: het gelijktijdig laten bewegen van een aantal objekten op het scherm. Met de bekende technieken kunnen we over het algemeen niet meer dan één of twee objekten laten bewegen, althans als we ons tot BASIC-instrukties beperken.

Gelukkig bestaat er een uitzondering op deze regel. Probeer het volgende programma maar eens uit.

```
10 CLS
20 PRINT AT 21,RND*31;"*";
30 SCROLL
40 GOTO 20
```

Door middel van een "SCROLL"-instruktie kunnen we alles op het scherm een regel naar boven laten bewegen en de snelheid waarmee dit gebeurt is hovendien onafhankelijk van de hoeveelheid informatie op het scherm.
De volgende vraag is: "Hoe kunnen we bij dit systeem bepaalde symbolen laten stilstaan? Wel, dan moeten we na iedere "SCROLL" het symbool dat stil moet blijven staan opnieuw met vaste coördinaten laten uitprinten. Voeg onderstaande regel toe aan het vorige programma:

```
35 PRINT AT 0,16;"V";
```

en er ontstaat een basisopzet, waarmee we bijvoorbeeld een ruimteschip door de sterrenhemel kunnen sturen!

Een ski-slalom spelletje
Een mooi voorbeeld van het werken met "SCROLL"-technieken in grafische programma's is het onderstaand ski-slalom programma.
(Bij het intoetsen van dit programma moet u in de regels 20, 2100, 3040, 4060 en 4110 een "kleiner dan" symbool onmiddellijk laten volgen door een "groter dan" symbool en mag u geen "niet-gelijk-aan" teken gebruiken!)

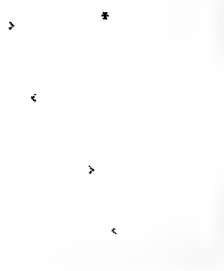
```
 10 RAND 0
 20 LET B$="<>"
 30 CLS
 40 DIM C(26,2)
 50 GOSUB 5000
 60 GOSUB 1000
 70 LET T=0
 80 LET P=0
 90 LET X=16
100 GOSUB 2000
110 GOTO 4000

1000 FOR I=1 TO 25
```

```
1010 LET C(I,1)=INT(RND*5)+5
1020 LET C(I,2)=INT(RND*25)+4
1030 NEXT I
1040 LET C(1,1)=21
1050 RETURN

2000 PRINT AT 21,C(1,2);">";
2010 LET K=1
2020 LET J=0
2030 LET L=0
2040 LET I=2
2050 LET S=0
2060 LET S=S+1
2070 LET T=T+1
2080 SCROLL
2090 GOSUB 3000
2100 IF S<>C(I,1) THEN GOTO 2060
2110 PRINT AT 21,C(I,2);B$(J+1);
2120 LET J=NOT J
2130 LET I=I+1
2140 IF I<26 THEN GOTO 2050
2150 FOR I=1 TO 23
2160 LET T=T+1
2170 SCROLL
2180 GOSUB 3000
2190 NEXT I
2200 RETURN

3000 LET A$=INKEY$
3010 IF A$="5" THEN LET X=X-1
3020 IF A$="8" THEN LET X=X+1
3030 PRINT AT 0,X;"*";
3040 IF T<>C(L+1,1) THEN RETURN
3050 LET T=0
3060 LET L=L+1
3070 LET K=NOT K
3080 IF NOT K AND (X-C(L,2))>0 THEN RETURN
3090 IF K AND (X-C(L,2))<0 THEN RETURN

3100 LET P=P+1
3110 PRINT "H"
3120 RETURN

4000 PRINT
4010 PRINT "U PASSEERDE ";P;" POORTEN"
4020 PRINT "PROBEERT U HET NOG EENS? J/N"
4030 INPUT A$
4040 PRINT A$
4050 IF A$(1)="N" THEN STOP
4060 IF A$(1)<>"J" THEN GOTO 4020
4070 PRINT "HET ZELFDE PARCOURS? J/N"
4080 INPUT A$
4090 PRINT A$
4100 IF A$(1)="J" THEN GOTO 70
4110 IF A$(1)<>"N" THEN GOTO 4070
4120 GOTO 60

5000 CLS
5010 PRINT
5020 PRINT TAB 8;"S K I   R U N"
5030 PRINT TAB 8;"-------------"
5040 PRINT
5050 PRINT "U KRIJGT EEN AFDALING"
5060 PRINT "MET 25 POORTJES"
5070 PRINT
5080 PRINT "U MOET < LINKS PASSEREN"
5090 PRINT "EN > RECHTS."
5100 PRINT
5110 PRINT "U KUNT NAAR LINKS EN RECHTS"
5120 PRINT "BEWEGEN MET DE PIJLTOETSEN."
5130 PRINT "(5 EN 8)"
5140 PRINT
5150 PRINT "DRUK EEN TOETS IN OM TE STARTEN"
5160 IF INKEY$="" THEN GOTO 5160
5170 CLS
5180 RETURN
```

De technieken, die in dit programma worden gehruikt, zijn vrij simpele voorbeelden van "SCROLL"-grafieken. Toch moeten we goed in de gaten houden waar alles zich op het scherm bevindt.
Door middel van subroutine 5000 worden enige instruktieteksten op het scherm geschreven.
Nadien wordt door middel van subroutine 1000 het parcours volgens de wetten der willekeur uitgezet. Er worden namelijk twee willekeurige getallen gegenereerd, de afstand tussen twee opeenvolgende poorten (gemeten in aantal "SCROLL's") en de horizontale positie van iedere poort. Het spel start als het programma een beroep doet op subroutine 2000, waardoor de eerste poort op het scherm wordt getekend en het parcours in de richting van de skiër (voorgesteld door een sterretje) wordt bewogen door middel van achtereenvolgende "SCROLL's".
Na een bepaald aantal "SCROLL's", voorgesteld door "C (2, 1)", wordt de volgende poort op het scherm gezet. Het moeilijke punt ontstaat na 21 "SCROLL's", want dan bereikt de eerste poort de skiër. Er zijn immers 22 bruik-

bare lijnen op het scherm! Op dat moment wordt de positie van de skiër bepaald en bekeken of hij zich aan de juiste kant van de poort bevindt. Volgens hetzelfde systeem wordt gecontroleerd hoeveel poorten de skiër passeert en hoe vaak hij aan de verkeerde kant van een poort zit. De details van deze programmering zijn erg eenvoudig (regels 2000 tot 3120), daar maken we geen woorden meer aan vuil.
Als u het programma korrekt intoetst, verschijnt het volgende beeld op het scherm.

```
S K I   R U N
-------------

U KRIJGT EEN AFDALING
MET 25 POORTJES

U MOET < LINKS PASSEREN EN > RECHTS

U KUNT NAAR LINKS EN RECHTS
BEWEGEN MET DE PIJLTOETSEN (5 EN 8)

DRUK EEN TOETS IN OM TE STARTEN
```



Gepagineerde grafiek
Met onze huidige kennis over de samenstelling van het beeldgedeelte van het geheugen en de ermee verband houdende variabelen, kunnen we een aantal aanvullende beelden creëren en door middel van "POKE"-instrukties naar believen een van deze beelden op het scherm laten uitprinten.

Met deze techniek kunnen we de ZX81 als het ware omvormen tot een "plaatjesalbum" en iedere "pagina" uit dit album zeer snel op het scherm zetten.
Bij deze werkwijze ontstaan gemakkelijk fouten, met als resultaat: een leeg scherm en een machine die niet meer naar bevelen luistert. Het uitschakelen van het apparaat is dan het enige dat er op zit, hetgeen uiteraard wel de vernietiging van de geheugeninhoud tot gevolg heeft.
Het principe is in essentie zeer eenvoudig, maar door de onvermijdelijke details wordt het geheel wel een beetje onoverzichtelijk. De ZX81 maakt gebruik van twee systeemvariabelen om op de hoogte te blijven van de schermbeelddetails. De eerste, "D-FILE", noteert waar het beeldgedeelte van het geheugen zich bevindt, de tweede, "DF-CC", kijkt waar het volgende symbool moet worden geprint.
Als u dus ergens in het geheugen een tweede beeldpagina opbouwt en u verandert deze variabelen op de juiste manier, dan wordt deze nieuwe beeldpagina op het scherm afgebeeld, in plaats van de originele. Zolang deze tweede pagina in beeld is, zullen alle "PRINT"-instrukties op deze pagina worden opgenomen en niet op de originele. U kunt terugschakelen naar de originele pagina, door het herstellen van de oorspronkelijke adressen in "D-FILE" en "DF-CC". Het enige waar u op moet letten is dat het "POKE"-en van nieuwe adressen in de twee genoemde variabelen alleen in de "FAST"-mode kan. De twee bytes moeten namelijk onmiddellijk na elkaar worden ge-"POKE"-d. Al zou de computer proberen tussen beide bewerkingen een beeld op het scherm te zetten, dan hebben de variabelen geen zinnige waarden en gaat de machine zich een beetje "uitzinnig" gedragen. In de "FAST"-mode wordt gedurende het verwerken van de nieuwe adressen van de variabelen het scherm niet uitgeschreven.
Om u enig idee te geven hoe dit systeem in de praktijk werkt, hebben we het volgende programma opgenomen. Zorg er echter wel voor, dat er geen fouten in staan, alvorens u het laat uitvoeren en onderneem geen aktie terwijl pagina 2 wordt gevormd!

```
10 LET E=100+PEEK 16412+256*PEEK 16413
20 GOSUB 1000
30 LET C=E
```

```
 40 LET D=E+1
 50 GOSUB 3000
 60 PRINT "SCHERM TWEE"
 70 FOR I=1 TO 100
 80 NEXT I
 90 LET C=A
100 LET D=B
110 GOSUB 3000
120 PRINT "SCHERM EEN"
130 FOR I=1 TO 100
140 NEXT I
150 LET C=E
160 LET D=E+1
170 GOSUB 3000
180 GOTO 70

1000 LET I=E
1010 FOR J=1 TO 24
1020 POKE I,11B
1030 LET I=I+1
1040 FOR K=1 TO 31
1050 POKE I,0
1060 LET I=I+1
1070 NEXT K
1080 NEXT J
1090 POKE I,11B
1100 RETURN

2000 LET A=PEEK 16396+256*PEEK 16397
2010 LET B=PEEK 16398+256*PEEK 16399
2020 RETURN

3000 GOSUB 2000
3010 FAST
3020 POKE 16396,C-INT(C/256)*256
3030 POKE 16397,INT(C/256)
3040 SLOW
3050 POKE 16398,D-INT(D/256)*256
3060 POKE 16399,INT(D/256)
3070 RETURN
```

Subroutine 1000 bouwt een nieuwe beeldpagina op vanuit het startadres "E". Deze nieuwe pagina is opgebouwd uit 25 "newline"-symbolen met 32 spatics ertussen.

Met subroutine 2000 worden de adressen van de beeldvariabelen opgeslagen in "A" en "B", terwijl subroutine 3000 de adressen van "C" en "D" in de schermvariabelen "POKE"-ed.

Het hoofdgedeelte van het programma start met het opzoeken van een vrij geheugendeel en het opbouwen van een nieuw beeldsysteem, respectievelijk door middel van de regels 10 en 20. Nadien wordt met behulp van de regels 30 tot en met 60 omgeschakeld naar de tweede beeldpagina en wordt "beeld twee" op het scherm geschreven.
Na een bepaalde vertraging schakelen de regels 90 tot en met 120 opnieuw de eerste pagina in en wordt deze ter bevestiging uitgeschreven.
Vervolgens worden de twee pagina's afwisselend op het scherm geschreven, waarbij de boodschap "beeld één" telkens een regel opschuift.
Deze methode om beelden in diverse pagina's op te slaan kan worden gebruikt voor het toepassen van speciale grafische effecten. Toch moeten we zeer voorzichtig zijn, daar steeds de mogelijkheid aanwezig is dat het programma wordt vernietigd en de computer op hol slaat. Vandaar dan ook, dat we deze techniek alleen aan gevorderde programmeurs aanbevelen.

Besluit

Nu u zoveel verschillende systemen voor het opbouwen van grafieken hebt leren kennen, moet u zonder meer in staat zijn zeer ingenieuze beeldkomposities op te bouwen. Zo zou u "SCROLL"-technieken kunnen mengen met "PEEK"-grafieken, maar denk daarbij aan de wijziging van de beeldopbouw! Als u nu echter denkt dat u met de beschreven technieken hetzelfde soort ingenieuze, op tekenfilm lijkende, beelden kunt opbouwen als mogelijk is met de moderne TV-spelcomputers, moeten wij u teleurstellen. De programmeertaal BASIC is daar gewoon niet snel genoeg voor!
Voor wie verder wil in computergrafiek heeft maar één keuze: overschakelen naar programmeren in machinecode, waarover in hoofdstuk 10 het een en ander wordt verteld.

5. HET ONTWERPEN VAN GROTE PROGRAMMA'S

Kleine programma's schrijven op een machine met een redelijk ruim geheugen is tamelijk gemakkelijk. In een klein programma gebeurt meestal niet zo erg veel en is het nog mogelijk alle opeenvolgende stappen in gedachten te houden tijdens het schrijven of nakijken van het geheel.
Het onvoorbereid beginnen aan het opstellen van een programma, kan een uitstekende methode zijn voor korte stukjes, het werkt niet meer als ingewikkelde problemen in computertaal moeten worden omgezet.
Het is immers onmogelijk alle stappen overzichtelijk in uw hoofd te houden, zonder dingen met elkaar te verwarren, en uiteraard is dat vragen om fouten.
De enige goede manier om gelijk welke omvangrijke klus aan te pakken is een systematische programmeermethode toe te passen en die consequent aan te houden.
Dit hoeft helemaal niet te betekenen dat het programmeren er saaier op wordt en dat alle pret wordt vervangen door een of ander dor systeem. Integendeel, het eindprodukt wordt er alleen maar beter op en wat geeft meer vreugde dan het ontwerpen van programma's waar men trots op kan zijn!

Het ontwerpen van een programma

Misschien heeft u wel eens gehoord dat een programma wordt opgebouwd aan de hand van een zogenaamde "flow chart" of "flow diagram". Vaak wordt zelfs beweerd dat het absoluut noodzakelijk is zo'n schema op te stellen alvorens men met het echte programmeren kan beginnen en dat die noodzaak alleen maar toeneemt als de problemen ingewikkelder worden.
Misschien heeft men u zelfs wijsgemaakt dat dit de enige goede programmeertechniek is en dat al het andere alleen maar flodderige programma's aflevert.
Uiteraard is dat volledig een kwestie van persoonlijke voorkeur en ook van gewoonte. Natuurlijk is het noodzakelijk dat u alvorens met het schrijven van een programma te beginnen, ruwweg een overzicht hebt van wat er allemaal te gebeuren staat, maar naar onze mening hoeft dit eerste inzicht niet zo gedetailleerd te zijn als noodzakelijk voor het tekenen van een "flowchart" (stroomdiagraz).
Ik persoonlijk, werk in eerste instantie steeds met een beschrijving van het programma in een mengelmoesje van Nederlands en BASIC. Nadien ga ik echt voor de computer zitten en het programma opstellen.

Als voorbeeld van deze techniek gaan we in deze paragraaf samen een programma opstellen, waarmee we kleine kinderen de beginselen van rekenkundige bewerkingen kunnen bijbrengen.
De eerste stap is nadenken over wat we het programma precies willen laten doen. Dit in gedachten op te stellen schema kan er ongeveer als volgt uit zien.

"Het programma moest eerst een vraag op het scherm schrijven en wachten op een antwoord. Als dit antwoord juist is, moet de gebruiker daarvan in kennis worden gesteld en moet de volgende vraag op het scherm verschijnen. Is het antwoord fout, dan moet ook dat worden gemeld en dezelfde vraag nogmaals op het scherm worden geschreven".

Noteer dat zelfs in deze eerste benadering al begrippen zitten, die we zo in BASIC kunnen overnemen, zoals "als" ("IF") en "schrijven" ("PRINT")!
U zou nu al het echte BASIC-programma kunnen opstellen, maar het is denkbaar dat u verkiest eerst toch nog een ietwat gedetailleerdere beschrijving, met bijvoorbeeld een preciese formulering van de vragen, op te stellen.
Deze manier voor het schrijven van computerprogramma's wordt de "stapsgewijze verfijning" genoemd, omdat er wordt begonnen met een zeer vaag idee en door dit idee telkens opnieuw uit te werken, ontstaat er een steeds gedetailleerdere beschrijving, die uiteindelijk resulteert in BASIC-regels. Na een aantal keren op die manier te hebben geprogrammeerd, zult

u tot de ontdekking komen dat dit in feite een zeer natuurlijke manier van werken is!
De tweede "verfijning" van het leren-rekenenspel zou er als volgt kunnen uitzien.

"Selekteer het soort bewerking: +; −; ×; :.
Genereer twee tamelijk kleine willekeurige getallen.
Schrijf het vraagstuk op het scherm en wacht op een antwoord.
Als het antwoord juist is, meldt dit dan aan de gebruiker en vervolg met het volgende vraagstuk.
Als het antwoord fout is, meldt dit dan aan de gebruiker en print dezelde vraag nogmaals op het scherm".

Het punt waarop u vindt dat deze "stapsgewijze verfijning" ver genoeg is gegaan om achter de computer te gaan zitten hangt natuurlijk volledig van uw eigen (on)zekerheid af! Het voordeel van dit systeem is, dat u het kunt aanpassen aan uw programmeerervaring en de moeilijkheidsgraad van het probleem. Daarnaast is er het niet te onderschatten psychologisch voordeel, dat er al van de eerste stap af iets op papier staat, hoe vaag ook. Vaak is dat nu net dat extra opstekertje, die ons over onze angst beenzet en ons doet besluiten toch maar voor het scherm te gaan zitten!

Het gebruik van subroutines

Hoewel de in de vorige paragraaf beschreven metbode van de "stapsgewijze verfijning" aardige resultaten oplevert, komen we er toch mee in de problemen als we te maken krijgen met zeer lange programma's. Als het aantal details toeneemt, worden we immers geconfronteerd met een immense hoeveelheid BASIC-instrukties en variabelen en zien we door de bomen het bos niet meer. In zo'n geval moeten we het geheel gaan onderverdelen in kleinere "brokjes". Dat kan meestal gemakkelijk, want een programma bestaat uit een aantal onderdelen die ieder een welomschreven taak uitvoeren. Zo kunnen we in het voorbeeld van het rekenprogramma drie duidelijk van elkaar gescheiden delen afbakenen:

- het deel dat de vragen opstelt;
- het deel dat de vragen op het scherm schrijft;
- het deel dat de antwoorden evalueert.

Ieder deel kan als een afzonderlijk programma worden behandeld en min of meer onafhankelijk van de overige worden uitgewerkt. De kunst bij het schrijven van grote programma's is dus het opzoeken van de diverse kleinere deelprogramma's. Soms zal blijken dat ook deze deelprogramma's in feite nog te groot zijn. Dan zullen we nog eens een onderverdeling moeten doorvoeren.
Het ligt voor de hand te veronderstellen dat dit principe van het opdelen van een programma in een aantal kleinere alles te maken heeft met het begrip "subroutine". De BASIC-instrukties "GOSUB" en "RETURN" kunnen worden gebruikt om een aantal instruktieregels om te zetten in een afzonderlijk, op zichzelf staand, programmadeel. Bij het voorbeeld van het rekenprogramma kunnen we veronderstellen dat er reeds twee subroutines bestaan, namelijk één startend bij regelnummer 1000 die het opstellen van de vragen voor zijn rekening neemt en één beginnend bij regel 2000, die de vragen op het scherm schrijft. De twee eerste regels van de tweede verfijning kunnen dan worden vervangen door:

```
10 GOSUB 1000
20 GOSUB 2000
```

Zo'n programma is in één oogopslag te overzien en het is tamelijk eenvoudig om te ontbouden waar we iedere subroutine voor willen gebruiken. Als we de "GOSUB's" zouden vervangen door de normale BASIC-regels voor het uitvoeren van de specifieke taken, zou het programma er zeer gecompliceerd uitzien. In principe zouden we ook het derde programmadeel door een subroutine kunnen vervangen. Toch is dat niet zo verstandig, omdat we in dit deel ofwel subroutine 2000 moeten laten berhalen (bij een fout antwoord) of het totale programma (bij een goed antwoord). We kunnen deze verwijzingen dan ook beter in de vorm van normale BASIC-regels als deel van het hoofdprogramma opbouwen.
Als we alles samenvatten, ontstaat bet volgende.

```
10 GOSUB 1000
20 GOSUB 2000
30 INPUT VRAAG
40 IF VRAAG=ANTWOORD THEN GOTO 10
50 PRINT "PROBEER HET NOG EENS"
60 GOTO 20
```

Een ander voordeel van het toepassen van subroutines is dat we ze in een hoofdprogramma kunnen gebruiken, ook al zijn ze nog niet geschreven! Als u ervan uitgaat dat die twee subroutines 1000 en 2000 te zijner tijd worden uitgewerkt, kunt u uw aandacht concentreren op het hoofdprogramma en later de subroutines één voor één onder handen nemen.
We kunnen dus een duidelijk verband leggen tussen het principe van de "stapsgewijze verfijning" en het gebruik van suhroutines. De details van de subroutines kunnen vrij vaag blijven tot ze echt aan de orde komen. Op dat moment kunt u zelfs besluiten dat het veel gemakkelijker is bepaalde subroutines weer op te splitsen in deel-subroutines ... en ga zo maar door tot alle programmastappen als eenvoudige subroutines zijn te beschrijven! Het gebruikmaken van nog ongeschreven subroutines bij het opstellen van het hoofdprogramma en het nadien op dezelfde manier behandelen van deze subroutines staat bekend onder de naam "modulair programmeren". U zult in de praktijk ontdekken, dat deze manier van programmeren zeer bevredigend en vruchtbaar is. Modulaire programma's zijn gemakkelijker te begrijpen en eenvoudiger aan te passen dan de gebruikelijke onoverzichtelijke lijst met BASIC-instrukties. Bovendien kunt u subroutines in allerlei hoofdprogramma's opnemen, zodat u als het ware niet telkens het wiel hoeft uit te vinden!

We kunnen nu de theorie in praktijk gaan brengen en trachten de nog ongeschreven subroutines van het leren-rekenen-spel uit te werken.

```
1000 LET A=INT(RND*10)
1010 LET B=INT(RND*10)
1020 LET O$="+-*/"(INT(RND*4)+1)
1030 LET ANTWOORD=VAL(STR$(A)+O$+STR$(B))
1040 RETURN

2000 CLS
2010 PRINT AT 10,5;A;" ";O$;" ";B;" ="
2020 PRINT AT 14,5;"WAT IS HET ANTWOORD ?"
2030 RETURN
```

Let op de manier waarop we in regel 1020 één van de vier basis-bewerkingen door het toeval laten uitkiezen en op het gebruik van de "VAL"-functie in regel 1030 voor het uitwerken van het antwoord.

Op het scherm verschijnt dan het volgende plaatje.

```
4 * 6 =

WAT IS HET ANTWOORD ?
```

Het BASIC-dialekt van de ZX81 heeft een extra eigenschap, waardoor het toepassen van subroutines nog overzichtelijker kan worden. Men kan de "GOSUB" en "GOTO"-instrukties op een andere dan de gangbare manier, namelijk gevolgd door een regelnummer, gebruiken. U kunt deze instrukties ook laten volgen door een meer algemene uitdrukking. Zo interpreteert de computer de regel "GOSUB 2 * 3 + 1" op dezelfde manier als "GOSUB 7". Op deze manier aangewend, kunnen we met dit grapje niet zo erg veel doen. Maar omdat een enkele variabele ook een uitdrukking is (zelfs de meest simpele vorm van een uitdrukking!) kunnen we bijvoorbeeld ook schrijven:

```
 5 LET GENQUESTION=1000
 6 LET PRINTQUESTION=2000
10 GOSUB GENQUESTION
20 GOSUB PRINTQUESTION
```

Nu wordt dit ogenschijnlijk nutteloze ideetje erg handig, want we kunnen suhroutines gaan benoemen, waardoor onze programma's niet alleen veel gemakkelijker zijn te lezen, maar ook eenvoudiger te veranderen.

De boodschap van deze paragraaf is simpel en duidelijk: gebruik subroutines! De volgende paragraaf is niet zo eenvoudig en u kunt hem met een gerust hart overslaan. We gaan ons namelijk bezig houden met een techniek, waar-

voor dezelfde waarschuwing geldt als voor de "gepagineerde grafiek" van hoofdstuk 4: voorbehouden aan experts!

Het uitgebreide "RETURN"-bevel

Er zijn gevallen denkbaar, waarbij het gebruik van een suhroutine erg handig kan zijn maar waarbij het om één of andere reden toch gemakkelijker is dat niet te doen. Zo werd het laatste gedeelte van het "leren-rekenen-programma" niet als een subroutine opgebouwd, omdat verschillende regels uit het hoofdprogramma resultaten opleverden, afhankelijk van de uitkomst van de hierin uitgevoerde test. De reden waarom het niet mogelijk is door de resultaten van een subroutine te laten bepalen naar welke regel van het hoofdprogramma moet worden gesprongen, is dat een suhroutine altijd eindigt met een "RETURN" en deze instruktie de computer altijd naar de regel, die volgt op de "GOSUB"-regel, stuurt.

In de meeste gevallen kunnen dit soort problemen worden opgelost voordat de subroutines worden uitgewerkt. Soms echter, blijkt pas hij het schrijven van een subroutine dat wat er verder in het hoofdprogramma moet gebeuren, afhankelijk is van de resultaten van de subroutine. Het zou dan zeer handig zijn als we de subroutine konden afsluiten met een "GOTO" regel nr....", waarbij het "regel nr...." zou worden hepaald door de resultaten van de suhroutine. In principe kan dit zonder dat er een foutmelding ontstaat, maar alleen als er slechts één subroutine tegelijkertijd wordt opgeroepen.

Het enige waarop we moeten letten is dat het "GOSUB"-gedeelte van het geheugen maar twee lokaties langer is dan noodzakelijk en dat deze reserve door het gebruiken van één subroutine zonder "RETURN" volledig kan worden opgebruikt. Als u dus van de ene suhroutine naar een andere wilt gaan met een "GOTO", dan kunt u er zeker van zijn dat er bij de volgende "RETURN"-regel grote problemen ontstaan!

We hehhen echt een "RETURN regel nr...."-instruktie nodig, waarmee we uit een subroutine naar een bepaalde regel kunnen gaan.

Wel moeten we duidelijk voor ogen houden, dat dit soort grapjes niet als normale BASIC-programma-technieken zijn te gebruiken, al kunnen we ze soms toepassen om een fout in het programma te herstellen.

Met onze huidige kennis van de geheugenopbouw van de computer is het niet zo moeilijk door middel van enige BASIC-regels het adres dat is opgeslagen in de "GOSUB"-stack te veranderen, zodat de volgende "RETURN" ons naar gelijk welke regel voert.

Als voorbeeld:

```
10 GOSUB 200
20 PRINT "HIER 20"
30 PRINT "HIER 30"
40 STOP

100 LET S=2+PEEK 16386+256*PEEK 16387
110 POKE S,L-INT(L/256)*256
120 POKE S+1,INT(L/256)
130 RETURN

200 PRINT "HIER 200"
210 LET L=30
220 GOTO 100
```

Door dit programma wordt eerst de mededeling "Hier 200" op het scherm geschreven (subroutine 200) en nadien de hoodschap "HIER 30". Als suhroutine 200 met een normale "RETURN" zou zijn afgesloten, dan zou de regel "HIER 20" ook op het scherm worden geschreven. Wat er nu gebeurt is dat de vier regels 100 tot en met 130 de geheugenlokatie opzoeken, waarin het regelnummer van de "RETURN" is opgeslagen, en dit nummer vervangen door de waarde van "L".

Door regel 210 wordt de waarde van "L" gelijk gesteld aan 30 en met regel 220 gaan we naar regel 100, waarin het net besprokene gebeurt. Vandaar dat de "RETURN" de computer naar regel 30 stuurt in plaats van naar regel 20 en de tekst "HIER 20" niet op het scherm verschijnt.

Het deelprogramma van de regels 100 tot en met 130 kan telkens worden gebruikt, als we na een subroutine naar een andere regel willen gaan dan waar een normale "RETURN" naar verwijst.

Programma-verzamelingen – menu's

Vaak kan een groot programma met sukses worden onderverdeeld in kleinere delen, waardoor het opstellen van het programma veel gemakkelijker wordt. Soms is het zo, dat we in grote programma's reeds bestaande kleinere

programma's kunnen verwerken. Deze kleinere delen kunnen zelfs los van elkaar zijn ontwikkeld en geschreven door verschillende programmeurs.
Door gebruik te maken van een zogenoemd "menu" kunnen ze evenwel overzichtelijk worden samengevoegd.

Als ieder deelprogramma een duidelijk gescheiden regelnummering meekrijgt, dan kunnen al die deelprogramma's los van elkaar worden ingetoetst. Wat er dan verder nog nodig is, is een extra programma, het menu, waarin de naam van de diverse deelprogramma's wordt vermeld en een verwijsmogelijkheid wordt opgenomen.
Zo kunnen de drie in hoofdstuk drie besproken "utilities" als volgt in de vorm van een menu worden gepresenteerd.

```
9000 CLS
9010 PRINT TAB(5);"U T I L I T I E S"
9020 PRINT AT 5,0
9030 PRINT "U KUNT KIEZEN UIT;"
9040 PRINT AT 10,0
9050 PRINT TAB 5;"(1) GEHEUGEN GEBRUIK"
9060 PRINT TAB 5;"(2) VARIABELEN LIJST"
9070 PRINT TAB 5;"(3) HERNUMMERING"
9080 PRINT TAB 5;"(4) STOP"
9090 PRINT AT 20,0;
    "TIK HET NUMMER VAN UW KEUZE IN"
9100 INPUT J0
9110 IF J0<1 OR J0>4 THEN GOTO 9090
9120 IF J0=1 THEN GOSUB 9200
9130 IF J0=2 THEN GOSUB 9300
9140 IF J0=3 THEN GOSUB 9800
9150 IF J0=4 THEN STOP
9160 GOTO 9000
```

Wel moeten de laatste regels van de drie "utilities" zo worden gewijzigd, dat de computer na het doorlopen van een "utility" terug naar het menu-programma gaat. Dat is niet zo moeilijk: de "STOP"-instrukties van de regels 9240 (utility 1), 9330 (utility 2) en 9900 (utility 3) worden vervangen door "RETURN"-regels. Het voordeel van zo'n menu is dat we niet meer alle startregels van de "utilities" hoeven te onthouden. Wilt u een "utility" gaan gebruiken, toets dan regel "GOTO 9000" in en het volgende verschijnt op het scherm.

```
U T I L I T I E S

U KUNT KIEZEN UIT;

(1) GEHEUGEN GEBRUIK
(2) VARIABELEN LIJST
(3) HERNUMMERING
(4) STOP

TIK HET NUMMER VAN UW KEUZE IN
```

Gebruikersvriendelijke programma's
Voor wie met programmeren begint, zal het een zorg zijn hoe een programma er in zijn uiteindelijke vorm uitziet. Als het maar werkt!

Toch zou het jammer zijn, als u dezelfde mentaliteit blijft behouden. De door u opgestelde programma's, hoe ingenieus ook bedacht, zijn dan negen van de tien keer alleen maar voor uzelf begrijpelijk! Misschien vindt u dat thans geen probleem, maar vergeet niet dat er ontelbare computerclubs bestaan.
En op het moment dat u lid wordt van zo'n club, zult u ontdekken hoe belangrijk het is dat uw programma's door anderen kunnen worden gebruikt.
Om maar niet te spreken over de mogelijkheid goed uitgewerkte programma's voor geld aan tijdschriften te verkopen!
Dat eist wel wat extra van de programmeur, namelijk een gevoel voor het opstellen van "gebruikersvriendelijke" programma's. Programma's, die ook voor de minst ervaren computergebruikers te verstaan zijn! Als u volgens dit systeem programma's wilt ontwerpen, moet u er altijd van uitgaan, dat het programma ook te begrijpen moet zijn voor mensen, die erg weinig van computers afweten en zelfs niet ingewijd zijn in het probleem, waarover het programma gaat.

Natuurlijk is het niet zo eenvoudig een kompleet overzicht te geven van de regels, die belangrijk zijn voor het ontwerpen van gebruikersvriendelijke programma's.

Onderstaande regels geven echter wél de basis aan, waaraan gebruikersvriendelijke programma's moeten voldoen:

- stel steeds klare en ondubbelzinnige vragen en vermijd het gebruik van vakjargon;
- kontroleer dadelijk het antwoord op iedere vraag en is het antwoord niet goed, herhaal dan de vraag;
- ga niet muggeziften over de juiste vorm van een antwoord: "J" is net zo goed als "JA";
- vul het scherm niet met te veel informatie en gebruik de "PAUSE" samen met "vervolg door het indrukken van een willekeurige toets" om het tempo van het programma aan te passen aan de gebruiker;
- voorkom door het toepassen van "PRINT AT"-instrukties, dat het scherm voortdurend met dezelfde vraag op dezelfde plaats wordt volgeschreven. Gebruik voor het uitprinten van uitvoerige lijsten met gegevens de instrukties "PRINT AT 21,0", "PRINT LIST" en "SCROLL";
- herhaal het antwoord op een vraag op de regel waar ook de vraag stond, zodat de gebruiker het antwoord kan terugvinden;
- neem de moeite een goede titelpagina op te stellen, met een korte introduktie over de mogelijkheden van het programma.

Kortom, het schrijven van goede gebruikersvriendelijke programma's is niet zo eenvoudig, maar de moeite van het proberen zeker waard!
Een goed voorbeeld van een uitstekend uitgewerkt programma is het statistisch programma in het volgende hoofdstuk.

Debugging

Het opzoeken van fouten in programma's is erg moeilijk uit een boekje te leren. De regels voor het opstellen van programma's zijn vrij eenvoudig aan te leren, maar de juiste manier om fouten uit programma's te halen is iets dat niet met een paar eenvoudige regels is te verwezenlijken. Daar doet men meestal jaren over.
Het is namelijk zo dat het "debuggen" van een programma iets is wat veel beter in de praktijk dan uit een of ander boek kan worden geleerd. Toch kunnen we wel een paar methodes aan de hand doen. Bovendien heeft de ZX81 enige handigheidjes, die nuttig zijn voor het opsporen van fouten.

Sommige fouten zijn gemakkelijk te ontdekken. Het programma doet niet wat we ervan verwachten, we laten de programmaregels uitprinten en de gedachte "hoe is het mogelijk dat ik dát niet heb gezien" flitst door ons hoofd. De fout ligt voor de hand en is in een mum van tijd hersteld. Moeilijker wordt het, als we na uren staren naar de reeks BASIC-instrukties niets verdachts kunnen vinden. Blijf in zo'n geval niet te lang naar het programma kijken! Dat helpt toch niet. Om een fout op te sporen moeten we kijken hoe het programma door de machine wordt uitgevoerd en komt dat overeen met de manier waarop het zou moeten.
Als u daar een tegenstrijdigheid in ontdekt, heeft u in ieder geval een aangrijpingspunt om de fout te lokaliseren, al wil dit nog niet zeggen dat we de fout dan ook direkt hebben.
Laten we eens kijken wat voor soort informatie interessant is voor het opsporen van fouten. Dat is niet zoveel als u misschien denkt. In feite hebben we maar in twee zaken interesse: de volgorde waarin de programmaregels worden uitgewerkt en de waarde die in de diverse variabelen is opgeslagen.
Sommige BASIC-dialekten hebben een "TRACE"-instruktie, die het nummer van de regel die wordt uitgewerkt op het scherm print. Hoewel men bij de ZX81 BASIC daar niet aan heeft gedacht, kunnen we wel de "STOP"- en de "CONTINUE"-instrukties toepassen. Als u wilt weten wat er op een bepaald moment gebeurt, voert u tijdelijk een "STOP"-instruktie in. De computer zal dan op dat punt stoppen en de bijbehorende rapportcode weergeven. Daar is ook het regelnummer in opgenomen, dus door het invoegen van een aantal "STOP"-codes in een programma kunt u toch een indruk krijgen van de volgorde waarin de diverse regels worden uitgevoerd. Na zo'n onderbreking kunt u door middel van "utility nummer 2" of met een "PRINT"-instruktie een idee krijgen over de in de variabelen opgeslagen waarden. U kunt zelfs met behulp van een "LET" de inhoud van een variabele wijzigen. Het enige wat u op deze manier niet kunt doen, is een extra regel tussenvoegen!
Als alles in orde is bevonden, kunt u door het intikken van "CONTINUE" het programma verder laten afwikkelen.
Het enige nadeel van deze methode is dat het gebruik van een "PRINT"-instruktie voor het

weergeven van de inhoud van de variabelen het scherm wist. Als u daarna verder werkt, blijft u dus met een incompleet scherm zitten!

Als u eenmaal weet dat er fouten in het programma zitten, kunnen ze er met dezelfde werkwijze tamelijk snel worden uitgehaald.
Een heel ander verhaal is het om er absoluut zeker van te zijn dat er geen verborgen fouten in een programma blijven staan. Eigenlijk bestaat hiervoor geen sluitende methode.
De enige, tijdrovende, manier is het programma met steeds weer andere begingegevens te laten lopen. Dit kan volgens bepaalde systemen, maar zelfs deze geven geen waterdichte garantie! Ze sturen verschillende programma's langs alle mogelijke wegen van de eerste naar de laatste regel.
Het beste wat u kunt doen, is het programma door iemand anders te laten testen. U, als ontwerper, kunt dagenlang bezig zijn met een bepaald programma en het foutvrij verklaren. Iemand anders heeft binnen dertig seconden soms toch nog een fout ontdekt! De meest voor de hand liggende reden is dat gebruikers van een programma op een andere manier met een programma omgaan dan de ontwerper.
Zij zullen beginnen met de makkelijkste delen van het geheel te testen, terwijl u, als ontwerper, eerder gaat kijken of er in de ingewikkeldste delen fouten zitten. Vergeet echter niet dat fouten net zo vaak in de voor de hand liggende delen van een programma voorkomen als in die stappen, waar u urenlang over hebt gezwoegd.
Als u ooit met zo'n vervelende ontdekking wordt geconfronteerd, moet u maar denken dat zelfs in de beroemdste programma's van de grootste software firma's soms toch nog fouten worden ontdekt, zelfs geruime tijd na het verschijnen op de markt!

6. GEGEVENS OP BAND

De ZX81 kan met behulp van de "SAVE"- en "LOAD"-instrukties rechtstrecks programma's op een band opnemen, maar geen "variabelen". Door het schrijven van bepaalde programma's zijn we echter wel in staat gegevensbestanden op cassetteband op te slaan.
In dit hoofdstuk komen de problemen, die daarbij om de hoek komen kijken, aan de orde en gaan we een tamelijk uitgebreid programma opstellen, waarbij we deze techniek zullen gebruiken.

Het principe van de bandopslag van de ZX81
Een van de mogelijkheden, die u waarschijnlijk het eerst hebt toegepast, is het "SAVE"-en en "LOAD"-en van een programma op een cassetteband. Heeft u een goede recorder dan werkt dit systeem feilloos, als u eenmaal de juiste stand van de volume- en toonregelingsknoppen hebt gevonden.
De volgende logische stap is uiteraard het opstellen van programma's, waarmee gegevensbestanden op de band kunnen worden gezet. Het probleem is dat de ZX81 slechts twee instrukties kent die het verkeer tussen computer en recorder regelen.

```
SAVE "PROGRAMMA-NAAM"
```

en

```
LOAD "PROGRAMMA-NAAM"
```

De meeste microcomputers kennen nog de volgende instruktie.

```
SAVE A
```

waarbij "A" staat voor de naam van het "array" dat op de band wordt gezet.
Het ontbreken van deze instruktie is dan ook één van de punten van kritiek op de ZX81. Een nadere studie van de werking van "SAVE" en "LOAD" bewijst echter dat deze instrukties heel wat meer in hun mars hebben dan op het eerste gezicht lijkt.
Door de instruktie "SAVE" wordt de inhoud van het geheugen tussen de lokatie met adres 16393 (dit adres ligt in de systeemvariabelenzone) en de lokatie waarvan we het adres terugvinden in "E-LINE", opgenomen op de band.
Dit geheugendeel omvat bijna alle systeemvariabelen, de volledige programmazone, de "display-file" en alle programmavariabelen. Wat op de band wordt gezet, geeft dus een momentopname van alle gegevens, die belangrijk zijn voor een lopend programma.
Als u op gelijk welk punt van een programma een "SAVE"-instruktie invoert, worden niet alleen alle BASIC-regels van dat programma opgenomen, maar ook de momentele inhoud van het scherm en alle variabelen. Als u deze informatie door middel van een "LOAD" opnieuw in het geheugen inleest, wordt die momentopname als het ware gereactiveerd en zal de computer het programma verder uitwerken vanaf de plaats waar u hem heeft gestopt.
Maar hoe valt het dan te verklaren dat ieder programma dat van de band in het geheugen wordt geladen, na de "RUN"-instruktie start vanaf het begin, zonder rekening te houden met wat er voorheen heeft plaatsgevonden? Het scherm is immers altijd leeg, er wordt geen rekening gehouden met een eerder "RUN"-bevel en alle variabelen worden op nul gezet.
Het antwoord op deze vraag is tweeledig. Op de eerste plaats zorgt een "RUN"-instruktie ervoor, dat het scherm wordt gewist en alle variabelen nul worden en ten tweede, dat het programma vanaf de eerste regel start. Het zal dus duidelijk zijn, dat we een "RUN" moeten voorkomen, als we een programma vanaf de band willen laden en het nadien verder willen laten lopen. Vandaar dat we in zo'n geval gebruik moeten maken van het "CONTINUE"-bevel.
U kunt dit kontroleren aan de hand van het volgende programma.

```
10 FOR I=1 TO 100
20 PRINT AT 21,0;I
30 SCROLL
40 NEXT I
```

Als u dit programma door middel van een "RUN"-instruktie door de machine laat uitvoe-

ren, zult u opmerken dat het scherm wordt volgeschreven met hele getallen van 1 tot en met 100.
Druk nu op een bepaald moment de "BREAK"-toets in en noteer het laatste op het scherm geschreven getal. Sluit nadien de recorder aan en zet hem in de opneem-stand.
Toets vervolgens:

```
SAVE "TEST"
```

in. Laat de recorder lopen en druk de "newline"-toets in. Als het opnemen klaar is, laat u het bandje terugspoelen en schakelt de computer uit, zodat het programma uit het geheugen is verdwenen. Schakel de machine weer in en geef het bevel:

```
LOAD "TEST"
```

Start de recorder in de weergeef-functie en geef een "newline". Als het programma in het geheugen zit, schakelt u de recorder uit, maar u gebruikt in plaats van een "RUN"- nu een "CONTINUE"-instruktie.
Het gevolg is, dat de machine gewoon verder gaat met het produceren van getallen alsof er niets is gebeurd. Het eerstvolgende getal uit het rijtje wordt op het scherm geprint!
Dit is een zeer interessante ontdekking, want door middel van deze "SAVE"- en "CONTINUE"-techniek kunt u het schrijven van een zeer lang programma op ieder gewenst moment onderbreken, het voorlopige resultaat op band bewaren, dit later weer in het geheugen inlezen en nadien verder gaan met het afwerken van het programma. U kunt zelfs door middel van een "GOTO"-instruktie naar een willekeurige regel van het programma gaan, zonder dat dit kwalijke gevolgen heeft voor de inhoud van de variabelenzone van het geheugen.
Toch heeft deze techniek een belangrijk nadeel. Hoewel het programma verder gaat vanaf het punt waar het is afgebroken, geldt hetzelfde niet voor de scherminhoud. Dat wordt veroorzaakt door het feit dat elke BASIC-regel, die wordt ingetoetst en direkt uitgevoerd (dus zonder regelnummer en niet deel uitmakend van een programma), het scherm wist.
Toets maar eens het volgende bevel in.

```
LET A=0
```

Na het indrukken van de "newline"-toets wordt het scherm gewist! Hetzelfde gebeurt, helaas, na het gebruik van een "SAVE"-instruktie. Alvorens de computer dit bevel uitvoert en het geheugen op de band opneemt, wordt het scherm gewist. En uiteraard start de machine na een "CONTINUE" met dit zelfde lege scherm!
Toch is er wel een oplossing voor dit probleem te verzinnen. Zowel "SAVE" als "LOAD" kunnen als normale programmaregels worden toegepast en dan hebben zij niet het wissen van het scherm tot gevolg. Bovendien heeft deze methode nog een tweede voordeel. Als u een programma, dat als gevolg van een in een programma opgenomen "SAVE"-instruktie op de band was gezet, weer in het geheugen laadt, zal na het laden dit programma gewoon verder gaan, dus zonder gebruik te maken van een "CONTINUE"!
De regel die als eerste wordt uitgewerkt is die, welke volgt op de "SAVE"-instruktie.

Deze eigenschap van onze computer is zeer nuttig, want we kunnen nu gedeeltelijk uitgewerkte programma's rustig op een band opnemen en nadien weer inlezen, zonder dat de oude scherminformatie verloren gaat! Bovendien kunnen we nu programma's ontwerpen, die zichzelf automatisch op een hepaald moment op de recorder opnemen!
Het volgende programma is een goed voorbeeld van hoe dat in de praktijk gaat.

```
10 FOR I=1 TO 100
20 PRINT AT 21,0;I
30 SCROLL
40 IF I=20 THEN SAVE "TEST"
50 NEXT I
```

Na een "RUN" zal de computer alle getallen van 1 tot en met 20 op het scherm gaan schrijven. Nadien zal het programma zichzelf op de band gaan opnemen! Wél moet u er natuurlijk voor zorgen, dat de recorder op dat moment loopt. Vandaar dat u er verstandig aan doet de recorder te starten op het moment dat de machine getal 15 op het scherm print.
Nadat de computer het getal 20 op het scherm heeft gezet, ziet u het bekende schermbeeld, dat altijd verschijnt als de machine bezig is met gegevens naar de recorder te zenden.
Nadat alles op hand staat, gaat de computer

door met het uitwerken van het programma, alsof er niets is gebeurd!
De getallen 21, 22, enz. worden op het scherm geschreven. Spoel de recorder terug en schakel de computer uit, zodat alle informatie over dit programma uit het geheugen is verdwenen. Laad nadien het op de band opgenomen programma op de gebruikelijke manier weer in de computer.
De machine gaat gewoon door met het uitprinten van getallen, maar bovendien verschijnt het oude schermbeeld weer op de heeldbuis!
Het gebruik van "SAVE" en "LOAD" als normale programmaregels heeft een nog niet genoemd voordeel.
U kunt niet alleen een "string" als naam voor een programma gebruiken, maar ook een volledige "string"-uitdrukking. Alle onderstaande regels zijn dus geldig.

```
SAVE A$
SAVE "PROG"+STR$(1)
LOAD A$+B$
```

Hoewel "SAVE" en "LOAD" zeer veelzijdig zijn, kunnen we de "SAVE"-instruktie niet opnemen in een subroutine.
Het "GOSUB"-gedeelte van het geheugen wordt immers niet op de band gezet en bijgevolg zal, na het herladen van het programma in het geheugen, de eerstvolgende "RETURN" er voor zorgen dat het programma stopt en een foutmelding op het scherm verschijnt.

Het opslaan van gegevens

Nu we weten hoe de "SAVE"-instruktie de inhoud van de variabelenzone van het geheugen op de hand zet, is het schrijven van een programma waarmee we gegevensbestanden kunnen opnemen en weergeven niet meer zo moeilijk.
Als voorbeeld behandelen we een statistisch programma, waarmee we cijferreeksen kunnen invoeren, kunnen manipuleren en veranderingen aanbrengen en bovendien op band kunnen opnemen voor later gebruik. De technieken die we in dit statistisch programma gaan toepassen, zijn algemeen bruikbaar voor het behandelen en op band opnemen van gelijk welke soort informatie.
We zullen dit programma opbouwen aan de hand van de technieken die we in het vorige hoofdstuk hebben bestudeerd. Met andere woorden: de methode van de "stapsgewijze verfijning" komt nu echt aan bod!
Allereerst moeten we dus een vrij vage indruk van het gewenste programma op papier zetten. Het zal duidelijk zijn, dat er een subroutine nodig is voor het inlezen van de cijferreeks in een "array". Daarnaast moeten we door middel van een tweede subroutine het getallenbestand bewerkbaar maken en door middel van een derde de gegevens opslaan op de band. Als de, op deze "SAVE" volgende instruktie een "GOTO begin programma" is, dan zal het programma na het opnieuw in het geheugen inschrijven vanaf het begin starten met de gegevens van de vorige "RUN" in het geheugen.
De gegevens die we op de band zetten, zijn een kombinatie van echte gegevens (de getallen die we intoetsen) en programmaregels. Bij de ZX81 kan dat ook niet anders. Programmeurs die gewend zijn met andere machines te werken, zullen dat een tamelijk vreemde eigenschap van onze ZX81 vinden. Zij zijn immers gewend aan het volledig gescheiden behandelen van programma's en gegevens.
Na even doordenken blijkt echter, dat er voor de manier waarop de ZX81 dit doet erg veel te zeggen valt.
Waarschijnlijk zullen andere construkteurs deze benadering in de toekomst ook gaan volgen. Het grote voordeel van het mengen van het programma en de gegevens, die in dat programma moeten worden verwerkt, is, dat men niet twee verschillende namen hoeft te gebruiken en te onthouden, namelijk één voor het programma en één voor het gegevensbestand. Het nadeel is dat het tamelijk moeilijk is gegevens van het ene naar het andere programma over te schrijven.
Het kost ook meer tijd om wijzigingen in het bestand aan te brengen. Men moet namelijk eerst het volledige programma van de band in het geheugen inlezen!
Toch kunnen we deze nadelen opheffen met behulp van "PEEK"- en "POKE"-instrukties en onze kennis van de geheugenstruktuur van de machine. Zo zou u gegevens van het ene naar het andere programma kunnen overschrijven door de volledige variabelenzone van het geheugen over te zetten naar de vrije-RAM. Bij het "LOAD"-en van een nieuw programma in de machine blijft dit geheugengedeelte immers

buiten schot en nadien kunnen we weer van de vrije-RAM terug naar het variabelendeel.

In de praktijk gaat dat echter allemaal niet zo eenvoudig en zolang we maar voldoende geheugenruimte ter beschikking hebben, kunnen we heel wat doen zonder dit zeer exotische soort spitsvondigheden te hoeven toepassen.
Het statistisch programma werkt goed, zolang we de hoeveelheid opgeslagen gegevens niet uitbreiden. Maar dat is nu net een van de voornaamste toepassingen van het werken met gegevensbestanden! Laat ons daarom eerst eens kijken wat het probleem is als we de hoeveelheid informatie willen vermeerderen. Alvorens we gegevens in de computer kunnen invoeren, moet het programma eerst een "array" reserveren om de gegevens te kunnen onderbrengen. Het zal duidelijk zijn, dat we dit "array" niet langer moeten maken dan noodzakelijk is voor het opslaan van de verwachte hoeveelheid gegevens. We willen geen tijd verliezen aan het op de band opslaan van ongebruikte "array"-elementen!

Een van de eerste vragen, die we het programma aan de gebruiker moeten laten stellen is dus hoeveel gegevens er te verwachten zijn. Aan de hand van het antwoord zal het programma een "array" met de juiste lengte vrij maken. Zolang het aantal ingevoerde gegevens in dat "array" past, is er niets aan de hand. Maar wat als we méér gegevens willen invoeren? We moeten dan de lengte van het "array" gaan aanpassen. We zouden dit kunnen doen door rechtstreeks ingrijpen in de variabelenzone van het geheugen. Gelukkig staat er ook een zeer eenvoudige programmeertechniek ter beschikking. Als het oorspronkelijke aantal variabelen gelijk was aan "N" en we willen "M" extra variabelen opnemen dan kunnen we een nieuwe "array" kreëren. Door de instruktie:

```
DIM E(N)
```

worden alle gegevens van de oorspronkelijke "array" (die we bijvoorbeeld "D" noemen) overgebracht naar een nieuwe "array" "E". Nadien kunnen we aan "D" een nieuwe, grotere, dimensie toekennen door:

```
DIM D(N+M)
```

Dit grapje kunnen we bij de ZX81 toepassen, omdat zijn "DIM"-instruktie iets andere eigenschappen heeft dan de standaard BASIC "DIM". Als men de afmetingen van een reeds bestaande "array" aanpast, wist de computer de bestaande "array" uit en vervangt deze door een nieuwe, waarvan de lengte niet gelijk hoeft te zijn aan die van de originele.
Nadat we de afmetingen van "D" hebben aangepast aan de nieuwe hoeveelheid gegevens, is het een koud kunstje de in "E" opgeslagen oude gegevens "N" in de eerste "N" elementen van de nieuwe "array" "D" te kopiëren en de gebruiker naar de nieuwe informatie "M" te vragen. Nadien mogen we echter niet vergeten het geheugendeel, door "E" gebruikt, weer vrij te maken door:

```
DIM E(1)
```

Natuurlijk is dit een vrij omslachtige manier om de hoeveelheid gegevens aan te passen, die bovendien traag is en veel geheugenruimte kost, maar alle andere beschikbare systemen zijn bij de ZX81 zeer gekompliceerd en niet zonder problemen in praktijk te brengen.

Het statistisch programma

Na deze inleiding kunnen we heginnen aan het ophouwen van het grootste programma uit dit boekje. We zullen veel van de reeds in de vorige hoofdstukken uitgelegde principes terugvinden en daarnaast kennis maken met enige nieuwe theorieën. Dit programma is meer dan alleen maar een demonstratie-voorbeeld; we kunnen er echt wat mee doen en bovendien kan het de basis vormen van zeer uitgebreide statistische programma's.
Wat biedt dit programma ons? Het "menu", behorende bij dit programma, geeft ons een zeer overzichtelijke samenvatting van alle mogelijkheden.

Enige toelichting.
Met het programma kunnen we gegevens in de computer invoeren (gang (1) van het menu); met deze gegevens als basis een aantal simpele statistische herekeningen uitvoeren (gang (5)), zoals: het bepalen van de minimum- en maximumwaarden, het gemiddelde, het verschil tussen minimum en maximum, de standaardafwijking, enz.

```
S T A T I S T I E K E N

(1) NIEUWE INFORMATIE
(2) MEK WILLEKEURIGE GETALLEN OP
(3) MANIPULEER MET GEGEVENS
(4) MEEM INFORMATIE OP
(5) BEREKEN STATISTISCHE WAARDEN
(6) TEKEN HISTOGRAM
(7) STOP

TIK HET GEWENSTE NUMMER IN
```

Daarnaast kunnen we de gegevens grafisch in de vorm van een histogram gepresenteerd krijgen (gang (6)). Dit is een grafiek, die met rechthoeken aangeeft hoe het totale gebied tussen de minimum- en maximumwaarden in een aantal intervallen is verdeeld en laat zien hoeveel van de ingevoerde waarden binnen ieder interval vallen. Uiteraard kunnen we door middel van gang (4) van het "menu" de ingevoerde gegevens op de band opnemen. Met behulp van gang (2) kunnen we de computer een aantal willekeurige getallen laten opwekken, die we als "gegevens" kunnen behandelen waarmee we het programma kunnen testen en de werking tonen zonder zelf een aantal gegevens te hoeven invoeren.
Tot slot geeft gang (3) van het menu ons de kans met de ingevoerde gegevens te manipuleren of, zoals de gebruikelijke "Engelse" vakterm luidt, te "editten".

Alvorens we het programma gaan bespreken, zullen we nog wat dieper ingaan op de verschillende faciliteiten en wel aan de hand van de tekst, die telkens op het scherm wordt geschreven bij de keuze uit dit "menu". Als we "edit data" kiezen. antwoordt de computer met het "edit-menu".

```
M A N I P U L E R E N  M E T  G E G E V E N S

(1) GEGEVENS BESTAAND
(2) WIJZIG GEGEVENS
(3) WIS GEGEVENS
(4) VOEG GEGEVENS TOE
(5) TERUG NAAR HOOFDMENU

TIK HET GEWENSTE NUMMER IN
```

Dit programma biedt dus vier "edit"-mogelijkheden. De eerste, "list data", zet alle gegevens op het scherm. Er ontstaat dan volgend staatje

```
LIJST BEGINT BIJ? 1
LIJST EINDIGT BIJ? 10

GEGEVEN WAARDE 1 = 2
GEGEVEN WAARDE 2 = 1
GEGEVEN WAARDE 3 = 5
GEGEVEN WAARDE 4 = 2
GEGEVEN WAARDE 5 = 1
GEGEVEN WAARDE 6 = 5
GEGEVEN WAARDE 7 = 4
GEGEVEN WAARDE 8 = 2
GEGEVEN WAARDE 9 = 3
GEGEVEN WAARDE 10 = 1

DRUK EEN TOETS IN VOOR VERVOLG
```

Met de "alter data"-gang kunnen we de waarde van een willekeurig gegeven veranderen. De computer vraagt eerst het volgnummer van het gegeven, print daarna de momentele waarde en vraagt nadien de nieuwe waarde.

```
WELK GEGEVEN MOET GEWIJZIGD? 4
HUIDIGE WAARDE = 2
NIEUWE WAARDE = 3

DRUK EEN TOETS IN VOOR VERVOLG
```

Met "delete data" kunnen we een deel van het bestand wissen. De computer vraagt in dat geval het volgnummer van het begin en het einde van het te wissen gebied.
De "add data"-gang van het "edit menu" zet het reeds genoemde systeem in werking, waarmee de cfmeting van de "array" aan de nieuwe hoeveelheid gegevens wordt aangepast. De machine vraagt eerst het aantal nieuwe in te voeren gegevens en gaat nadien aan de slag. Omdat dat nogal wat tijd vergt, print de computer tussendoor enige mededelingen op het scherm, zodat we er zeker van zijn dat hij niet in slaap is gevallen!

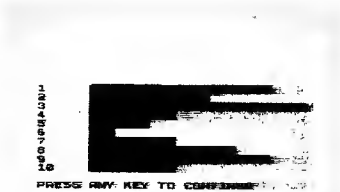
```
HOEVEEL GEGEVENS WORDEN TOEGEVOEGD? 10
WACHT,TERWIJL IK RUIMTE MAAK.
BIJNA KLAAR
KLAAR VOOR NIEUWE GEGEVENS
GEGEVEN WAARDE 11 =
```

Kiezen we "calculate statistics" uit het hoofd-"menu", dan berekent de computer de reeds genoemde statistische waarden en zet ze overzichtelijk op het scherm:

```
AANTAL NAARDEN=40
MAXIMUM=10
MINIMUN=1
REEKS=9
GEMIDDELDE=8
VERSCHIL=18.051282
STANDAARD AFWIJKING=4.24868

DRUK EEN TOETS IN VOOR VERVOLGEN
```

Het intoetsen van gang (6) uit het hoofd"menu" levert het onderstaande prentje op.

```
HOEVEEL INTERVALLEN ?10
MAXIMUM WAARDE= 11
MINIMUM WAARDE= 1
```



De computer vraagt het aantal intervallen, de minimum- en maximumwaarden en produceert daarmee een histogram, waaruit onmiddellijk valt af te leiden hoe de waarden van de ingevoerde gegevens over het totale gebied zijn verdeeld.
Gang (4) van het hoofd-"menu", tenslotte, produceert onderstaande tekst op het beeldscherm:

```
DRUK EEN TOETS IN VOOR VERVOLGEN

STOP EEN BLANKO CASSETTE IN
DE RECORDER

WELKE NAAM GEEFT U AAN HET
PROGRAMMA ?
TEST

DRUK SPEEL EN OPNEEMTOETSEN IN
DRUK DAARNA EEN WILLEKEURIGE TOETS IN
OP HET TOETSENBORD
```

Zoals u ziet, een zeer nuttig programma!
Nadat het programma van de cassette in het geheugen is geladen, moeten we ofwel de gegevens intoetsen, ofwel de computer zélf een aantal willekeurige getallen laten berekenen. Nadien kunnen we steeds vanuit het hoofd-"menu" een van de faciliteiten kiezen, er mee werken, terug gaan naar het hoofd-"menu" en dezelfde of een andere faciliteit kiezen. Als we de gegevens en het programma op band willen zetten, volstaat het de juiste gang uit het hoofd-"menu" te kiezen.
Een laatste opmerking alvorens we het programma presenteren. Het programma stopt niet automatisch, maar na het op de band zetten van alle gegevens kunnen we de machine met een gerust hart uitschakelen. Als u nieuwe gegevens in het bestand wilt invoeren, is het voldoende het programma weer in het geheugen te laden met het gebruikelijke bevel "LOAD", gevolgd door de gekozen programmanaam. Het programma start nadien automatisch en alle gegevens staan weer ter beschikking.
Zo, en nu het programma, dat uit niet minder dan 259 regels bestaat!

```
 10 REM STATISTISCH PROGRAMMA

500 CLS
510 PRINT TAB 5;"S T A T I S T I E K E N "
520 PRINT AT 6,0
530 PRINT "(1) NIEUWE INFORMATIE"
540 PRINT "(2) NEK WILLEKEURIGE GETALLEN OP"
550 PRINT "(3) MANIPULEER MET GEGEVENS"
560 PRINT "(4) NEEM INFORMATIE OP"
570 PRINT "(5) BEREKEN STATISTISCHE WAARDEN"
580 PRINT "(6) TEKEN HISTOGRAM"
590 PRINT "(7) STOP"
600 PRINT AT 21,0;"TIK HET GEWENSTE NUMMER IN";
610 INPUT SEL
620 IF SEL= 1 THEN GOSUB 3000
630 IF SEL= 2 THEN GOSUB 1000
640 IF SEL= 3 THEN GOSUB 4000
```

```
650 IF SEL= 4 THEN GOTO 1500
660 IF SEL= 5 THEN GOSUB 5500
670 IF SEL= 6 THEN GOSUB 6000
680 IF SEL= 7 THEN STOP
690 GOTO 500


1000 CLS
1010 PRINT "NILLEKEURIGE GETALLEN"
1020 PRINT "AANTAL?";
1030 INPUT N
1040 PRINT N
1050 DIM D(N)
1060 PRINT AT 3,0;"GEBROKEN OF HELE GETALLEN? G/H";
1070 INPUT A$
1080 IF A$(1)<>"G" AND A$(1)<>"H" THEN GOTO 1060
1090 PRINT A$
1100 LET T=0
1110 IF A$="H" THEN LET T=1
1120 PRINT AT 4,0;"LAAGSTE WAARDE";
1130 INPUT L
1140 PRINT L
1150 PRINT "HOOGSTE WAARDE";
1160 INPUT H
1170 PRINT H
1180 IF H>L THEN GOTO 1210
1190 PRINT "HOOGSTE<LAAGSTE"
1200 GOTO 1120
1210 FOR I=1 TO N
1220 LET D(I)=RND*(H-L+T)+L
1230 IF T=1 THEN LET D(I)=INT D(I)
1240 PRINT AT 20,0;"GEGEVEN WAARDE ";I;" = ";D(I)
1250 SCROLL
1260 NEXT I
1270 GOTO 8900


1500 CLS
1510 PRINT "STOP EEN BLANKO CASSETTE IN"
1520 PRINT "DE RECORDER"
1530 PRINT
1540 PRINT "WELKE NAAM GEEFT U AAN HET"
1550 PRINT "PROGRAMMA?";
1560 INPUT A$
1570 PRINT A$
1580 PAUSE 25
1590 PRINT "DRUK SPEEL EN OPNEEMTOETSEN IN"
1600 PRINT "DRUK DAARNA EEN WILLEKEURIGE TOETS IN"
1610 PRINT "OP HET TOETSENBORD"
1620 IF INKEY$="" THEN GOTO 1620
1630 PRINT "TOT ZIENS"
1640 PAUSE 50
1650 CLS
1660 SAVE A$
1670 GOTO 10


2000 LET M=0
2010 LET S=0
2020 LET L=D(1)
2030 LET H=L
2040 FOR I=1 TO N
2050 LET M=M+D(I)
2060 IF L>D(I) THEN LET L=D(I)
2070 IF H<D(I) THEN LET H=D(I)
2080 NEXT I
2090 LET M=M/N
2100 FOR I=1 TO N
2110 LET S=S+(D(I)-M)*(D(I)-M)
2120 NEXT I
2130 LET S=S/(N-1)
2140 RETURN


2500 CLS
2510 PRINT "AANTAL WAARDEN= ";N
2520 PRINT "MAXIMUM= ";H
2530 PRINT "MINIMUM= ";L
2540 PRINT "REEKS= ";H-L
2550 PRINT "GEMIDDELDE= ";M
2560 PRINT "VERSCHIL= ";S
2570 PRINT "STANDAARD AFWIJKING= ";SQR(S)
2580 GOTO 8900


3000 CLS
3010 PRINT "INVOER GEGEVENS? "
3020 PRINT "HOEVEEL WAARDEN ? ";
3030 INPUT N
3040 PRINT N
3050 DIM D(N)
3060 FOR I=1 TO N
3070 PRINT AT 21,0;"WAARDE ";I;" = ";
3080 INPUT D(I)
3090 PRINT D(I)
3100 SCROLL
3110 NEXT I
3120 PRINT "INVOER GEGEVENS VOLTOOID"
3130 SCROLL
3140 GOTO 8900


4000 CLS
4010 PRINT TAB 5;
     "M A N I P U L E E R  M E T  G E G E V E N S"
4020 PRINT AT 5,0
```

```
4030 PRINT "(1) GEGEVENS BESTAND"
4040 PRINT "(2) NIJZIG GEGEVENS"
4050 PRINT "(3) NIS GEGEVENS"
4060 PRINT "(4) VOEG GEGEVENS TOE"
4070 PRINT "(5) TERUG NAAR HOOFDMENU"
4080 PRINT AT 21,0;"TIK HET GEWENSTE NUMMER IN"
4090 INPUT ED
4100 IF ED=1 THEN GOSUB 4200
4110 IF ED=2 THEN GOSUB 4500
4120 IF ED=3 THEN GOSUB 4600
4130 IF ED=4 THEN GOSUB 4800
4140 IF ED=5 THEN RETURN
4150 GOTO 4000

4200 CLS
4210 PRINT "LIJST BEGINT BIJ ?";
4220 INPUT L
4230 PRINT L
4240 PRINT "LIJST EINDIGT BIJ (-1 = EIND)?";
4250 INPUT H
4260 IF H<0 THEN LET H=N
4270 PRINT H
4280 IF L>H THEN GOTO 4200
4290 IF L>N OR H>N OR L<1 OR H<1 THEN GOTO 4200
4300 FOR I=L TO H
4310 PRINT AT 20,0;"GEGEVEN NAARDE ";I;" = ";D(I)
4320 SCROLL
4330 IF INT((I-L+1)/20)*20=(I-L+1)THEN GOSUB 8900
4340 NEXT I
4350 GOTO 8900

4500 CLS
4510 PRINT "WELK GEGEVEN MOET GENIJZIGD?";
4520 INPUT I
4530 IF I<1 OR I>N THEN GOTO 4500
4540 PRINT I
4550 PRINT "HUIDIGE NAARDE= ";D(I)
4560 PRINT "NIEUWE WAARDE= ";
4570 INPUT D(I)
4580 PRINT D(I)
4590 GOTO 8900

4600 CLS
4610 PRINT "WISSEN VANAF ";
4620 INPUT L
4630 PRINT L
4640 PRINT "EINDIGEND BIJ ";
4650 INPUT H
4660 PRINT H
4670 IF H<L THEN GOTO 4600
4680 IF H>N OR H<1 OR L>N OR L<1 THEN GOTO 4600
4690 PRINT
4700 PRINT "WISSEN VANAF ";L;" TOT ";H
4710 PRINT "IS DIT CORRECT J/N ";
4720 INPUT A$
4730 PRINT A$
4740 IF A$(I)<>"J" THEN RETURN
4750 FOR I=H+1 TO N
4760 LET D(L+I-H-1)=D(I)
4770 NEXT I
4780 LET N=N-H+L-I
4790 RETURN

4800 CLS
4810 PRINT "HOEVEEL GEGEVENS NOKDEN TOEGEVOEGD? ";
4820 INPUT M
4830 PRINT M
4840 DIM E(N)
4850 PRINT "NACHT,TERNIJL IK RUIMTE MAAK."
4860 FOR I=1 TO N
4870 LET E(I)=D(I)
4880 NEXT I
4890 PRINT "BIJNA KLAAR"
4900 DIM D(N+M)
4910 FOR I=I TO N
4920 LET D(I)=E(I)
4930 NEXT I
4940 PRINT "KLAAR VOOR NIEUWE GEGEVENS"
4950 DIM E(1)
4960 FOR I=N+1 TO N+M
4970 PRINT AT 21,0;"GEGEVEN NAARDE ";I;" = ";
4980 INPUT D(I)
4990 PRINT D(I)
5000 SCROLL
5010 NEXT I
5020 PRINT "GEGEVENS INVOER VOLTOOID"
5030 SCROLL
5040 LET N=N+M
5060 GOTO 8900

5500 CLS
5510 PRINT "REKENEN"
5520 GOSUB 2000
5530 GOTO 2500

6000 CLS
6010 PRINT "HOEVEEL INTERVALLEN? ";
6020 INPUT M
6030 PRINT M
6040 PRINT "MAXIMUM NAARDE= ";
```

```
6050 INPUT H
6060 PRINT H
6070 PRINT "MINIMUM WAARDE= ";
6080 INPUT L
6090 PRINT L
6100 IF H<L THEN GOTO 6000
6110 LET D=(H-L)/M
6120 GOSUB 7000
6130 FOR I=1 TO M
6140 PRINT AT 21,0;INT(L*100)/100;TAB 6;
6150 IF H(I)=0 THEN GOTO 6190
6160 FOR J=1 TO H(I)/F*25
6170 PRINT "[";
6180 NEXT J
6190 SCROLL
6200 LET L=L+D
6210 NEXT I
6220 SCROLL
6230 GOTO 8900

7000 DIM H(M)
7010 FOR I=1 TO N
7020 LET J=(D(I)-L)/(H-L)*M+1
7030 LET J=INT J
7040 IF J<1 OR J>M THEN GOTO 7060
7050 LET H(J)=H(J)+1
7060 NEXT I
7070 LET F=0
7080 FOT I=1 TO M
7090 IF F<H(I) THEN LET F=H(I)
7100 NEXT I
7110 RETURN

8900 PRINT AT 21,0;
     "DRUK EEN TOETS IN VOOR VERVOLGEN"
8910 IF INKEY$="" THEN GOTO 8910
8920 RETURN
```

Uiteraard is dit programma veel te lang om alle details uitvoerig te bespreken. Het geheel is zeer overzichtelijk met een groot aantal subroutines opgebouwd. Een lijstje van deze subroutines en hun taak, vormt een uitstekende plattegrond om de weg in het geheel te vinden.
De meeste subroutines zijn vrij eenvoudig en gemakkelijk te begrijpen. Als we ons alles voor de geest halen wat we tot nu toe hebben geleerd, zullen ook de iets ingewikkeldere delen duidelijk worden.
Toch enige opmerkingen.

regelnummers	taak van de subroutine
10	naam van het programma
500- 690	het hoofd-"menu"
1000-1270	berekenen van willekeurige getallen
1500-1670	het "SAVE"-en van programma en gegevens
2000-2140	de statistische berekeningen
2500-2580	het uitprinten van de berekeningen
3000-3140	de gegevensinvoer
4000-4150	het edit-"menu"
4200-4350	edit-gang (1), uitprinten van gegevens
4500-4590	edit-gang (2), gegevens wijzigen
4600-4790	edit-gang (3), gegevens wissen
4800-5060	edit-gang (4), uitbreiden "array"
5500-5530	berekenen en printen statistische grootheden
6000-6230	tekenen van het histogram
7000-7110	berekenen inhoud van de intervallen
8900-8920	wachten op toetsdruk om het programma te vervolgen.

De regels 1500-1670 transporteren de gegevens en het programma naar de band. Er wordt geen gebruik gemaakt van een subroutine, maar van een "GOTO" in regel 650. Zoals we weten is dat noodzakelijk, omdat het automatische "RUN"-nen van het programma niet werkt als de "SAVE" in een subroutine is opgenomen. In subroutine 4200 zorgt regel 4330 ervoor, dat de machine na iedere 20 ge-"SCROLL"-de schermregels pauzeert. Door middel van suhroutine 4800 worden nieuwe gegevens aan het bestand toegevoegd, volgens het reeds besproken systeem.
De technieken die in de subroutines 6000 en 7000 worden gebruikt voor het opbouwen van het histogram zijn nieuw. Subroutine 7000 telt het aantal waarden dat binnen ieder interval valt door de instruktie van regel 7020. Als u zich voorstelt dat de intervallen vanaf 1 worden genummerd van links naar rechts, dan zal met de vergelijking het nummer worden berekend van het interval, waarbinnen een bepaalde waarde valt. Met dit intervalnummer wordt nadien de waarde van de bijbehorende "H"-waarde met één verhoogd.
Zodoende vinden we in "H(I)" het aantal waarden dat binnen interval "I" valt. Door middel van subroutine 6000 print de machine voor iedere "H(I)"-waarde het juiste aantal vierkantjes op het scherm.

Er kunnen maximaal 25 blokjes op het scherm worden getekend. Dit is dus de lengte van de langste kolom en voor iedere eenheid in een "H"-waarde wordt bijgevolg 25/"F" blokjes op het scherm getekend. Hierbij staat "F" voor het aantal waarden in de langste kolom. Voor iedere waarde van "H" moeten dus "H(I)*25/F" blokjes worden getekend, hetgeen door de regels 6160 tot en met 6180 wordt verzorgd.
Op deze vrij ingewikkelde manier zorgen we ervoor, dat de kolommen van het histogram nooit kunnen "overlopen". Er zijn een heleboel details die we onbesproken hebben gelaten. De beste manier om het programma volledig te leren beheersen, is er in de praktijk mee te werken, het te verbeteren en aan te passen aan uw specifieke wensen.
Dat aanpassen gaat, door de modulaire opbouw, vrij eenvoudig. Enige suggesties: het toevoegen van een tabel, waarin de lengte van alle kolommen van het histogram wordt gegeven; het programma zo wijzigen dat met meer dan één kolom kan worden gewerkt; het kiezen van de kolom waarop we de statistische berekeningen willen toepassen en (alleen voor de gevorderden!) het uitbreiden met subroutines, waarmee we de diverse kolommen met elkaar kunnen vergelijken en spreidingsdiagrammen kunnen opstellen. Enige basiskennis van statistisch rekenen is hiervoor noodzakelijk!

7. GETALLEN FORMEREN

Hoewel het overdreven zou zijn om het een ernstige tekortkoming te noemen, heeft de ZX81 als nadeel dat er geen systemen bestaan om vat te krijgen op de manier waarop getallen op het scherm worden geschreven. Als u het bevel "PRINT A" geeft, kunt u niet voorspellen hoeveel cijfers er voor en achter de komma zullen verschijnen en nog minder waar die komma zal worden geprint. Het enige wat u kunt doen, is door middel van een "PRINT TAB"- of "PRINT AT"-instruktie de schermlokatie van het eerste cijfer van het getal vastleggen. Misschien vraagt u zich af, wat er zo belangrijk is aan het kunnen ingrijpen in de manier waarop de computer getallen uitschrijft. Toch ligt het antwoord op die vraag voor de hand: we kunnen er meer overzichtelijke en minder misleidende resultaten mee verkrijgen.
Als u bijvoorbeeld een heleboel getallen onder elkaar hebt geprint en ze nadien wilt lezen, is dat veel gemakkelijker als alle komma's onder elkaar staan! Verder is het onzin om de machine het resultaat van een berekening tot 10 cijfers achter de komma te laten uitprinten, want de herekening is toch maar tot drie cijfers na de komma nauwkeurig.
Kortom, werken met getallen en cijfers eist ingrijpen door de programmeur. Hoewel de ZX81 geen instrukties, om de manier waarop getallen op het scherm worden geschreven volledig te kunnen sturen, ter beschikking heeft, kunnen we toch vergelijkbare resultaten krijgen door een aantal subroutines op te stellen.

Afronden en inkorten

Het gevaar van te veel cijfers na de komma als resultaat van een berekening is reeds genoemd. De suggestie wordt gewekt dat de berekening tot het laatste cijfer toe nauwkeurig is, wat niet per definitie zo hoeft te zijn! Toch komt het vrij vaak voor dat de computer als resultaat van een vrij simpele berekening voor twee getallen met een paar cijfers na de komma, een getal levert met het maximale aantal cijfers waartoe de machine in staat is na de komma. Bovendien is het een niet algemeen bekend feit, dat een digitale computer berekeningen slechts met een beperkte nauwkeurigheid uitvoert en dat de uitkomst steeds minder in overeenstemming is met de gegevens die we in de berekening hebben gebruikt.
In feite zouden we dus regels moeten opstellen, waaruit we voor ieder soort berekening precies kunnen afleiden hoe nauwkeurig de resultaten zijn en hoeveel cijfers na de komma nog betrouwbaar. Maar daar zijn nauwelijks exacte regels voor te geven! De beste benadering is uit te gaan van de praktische situatie en te bekijken waarvoor we een resultaat gaan gebruiken. Zo is het kompleet waanzin om het geschatte brandstofverbruik van een centrale verwarming als volgt door de computer te laten uitprinten.

```
NODIGE AANTAL LITERS OLIE= 102.34983723
```

Waarschijnlijk zal uw brandstofleverancier u alleen maar meewarig aankijken als u 102.34983723 liter olie bestelt! Kortom, we mogen programmaresultaten niet als op zichzelf staande gegevens opvatten, maar als grootheden die de mens ten dienste staan.
Om nog even terug te komen op dat brandstofverbruik. Het is al even zinloos het brandstofverbruik van diverse installaties met elkaar te vergelijken in de vorm van een tabel, uitgeprint met getallen tot op 10 cijfers na de komma. Het resultaat zal er alleen maar onoverzichtelijk door worden.
Kortom, we moeten getallen gaan inkorten. Als we gebruik maken van de "INT"-functie gaat dat zeer eenvoudig.
Toets het volgende programma in.

```
 10 FOR N=1 TO 10
 20 LET V=RND
 30 PRINT "GETAL= ";N;TAB 15;
 40 GOSUB 1000
 50 PRINT
 60 NEXT N
 70 STOP

1000 LET DIG=10**N
1010 PRINT INT (V*DIG)/DIG;
1020 RETURN
```

Na een "RUN" print dit programma een reeks getallen op het scherm, met steeds meer cijfers na de komma.
Suhroutine 1000, die verantwoordelijk is voor het inkorten van de getallen, kan in elk programma worden toegepast. "V" is het te printen getal en "N" het gewenste aantal cijfers na de komma. We kunnen de werking van dit programma het eenvoudigst toelichten aan de hand van een voorbeeld. Stel dat "V" gelijk is aan 0,123 en "N" gelijk is aan 2. Uit regel 1000 volgt, dat "DIG" gelijk is aan 10**2, of met andere woorden gelijk aan het kwadraat van 10, dus 100. Het vermenigvuldigen van "V" met 100 levert 12,3 op, de "INT"-functie houdt alleen het gehele deel, dus 12, over. Deling van dit resultaat door "DIG" geeft de oorspronkelijke waarde terug aan "V", maar nu ingekort tot het gewenste aantal decimalen, dus als 0,12.
Wél moeten we er rekening mee houden dat hoewel "N" het maximale aantal cijfers na de komma bepaalt, het best mogelijk is dat een resultaat met minder cijfers na de komma op het scherm verschijnt.
Dat blijkt bijvoorheeld al uit het resultaat van het vorige programma, waar in principe het aantal cijfers na de komma van 1 tot en met 9 zou moeten oplopen, maar waar soms minder decimalen zijn afgedrukt.

```
DIGITS=1      0.5
DIGITS=2      0.71
DIGITS=3      0.354
DIGITS=4      0.3097
DIGITS=5      0.23226
DIGITS=6      0.42221
DIGITS=7      0.66645B1
DIGITS=8      0.98474121
DIGITS=9      0.85560608
DIGITS=10     0.17060852
```

Het nadeel van deze subroutine is dat zij langc getallen inkort door het "afhakken" van een aantal cijfers. terwijl het over het algemeen gebruiklijk is dat er wordt afgerond. Bij afronden kijkt men naar het eerste cijfer dat wegvalt. Is dat 5 of groter, dat wordt het vorige cijfer met één eenheid verhoogd. Het getal 0,126 wordt volgens de besproken subroutine "afgehakt" tot 0,12, maar zou volgens de regels van het afronden 0,13 worden.
We kunnen subroutine 1000 vrij eenvoudig aanpassen aan de wetten der afronding. Regel 1010 wordt dan:

```
1010 PRINT INT (V*DIG+.5)/DIG;
```

Als u hetzelfde getallenvoorbeeld op deze formule loslaat, zult u het toegepaste afrondingsprincipe snel begrijpen.

Het uitlijnen van de komma

Als u het vorige programma een aantal keren laat lopen, zult u meestal niet de mooie driehoek zien, waartoe het programma in principe in staat is. Vaker zult u een resultaat verkrijgen dat er vrij slordig uitziet.

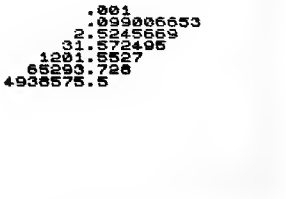
```
DIGITS= 1      0.4
DIGITS= 2      0.25
DIGITS= 3      0.127
DIGITS= 4      .0808
DIGITS= 5      .02575
DIGITS= 6      0.223493
DIGITS= 7      .0977454
DIGITS= 8      0.35104859
DIGITS= 9      .068214123
DIGITS= 10     0.12482941
```

Om dit schoonheidsfoutje te verbeteren, volstaat het een subroutine te ontwerpen, waarmee we de positie van de komma op het scherm kunnen vastleggen, natuurlijk in kombinatie met het afronden van te veel cijfers na de komma.
Toets het volgende programma in.

```
10 LET M=7
20 INPUT V
30 PRINT TAB 10;
40 GOSUB 3000
50 LET V=V*100*RND
60 GOTO 30
```

```
3000 PRINT "          "(1 TO M-(V>1)*INT(LN
  V/LN 10)+(V<.1));V
3010 RETURN
```

Na een "RUN" verschijnt het volgende resultaat op het scherm.



Dit programma lijkt tamelijk gekompliceerd, maar is toch snel te begrijpen als we het gaan ontleden in de diverse delen. Met de regels 10 tot 60 worden testgetallen opgewekt, die we in subroutine 3000 bewerken. Met de variabele "M" kunnen we het aantal printposities voor de komma vastleggen. "V" is uiteraard weer het getal waarop we de subroutine gaan toepassen. Om er zeker van te zijn dat voor ieder getal de komma op dezelfde plaats op het scherm verschijnt, is het noodzakelijk een variabel aantal spaties voor het echte getal op te nemen. Als we bijvoorbeeld een maximaal aantal printposities van 5 in "M" vastleggen en we moeten het getal 22,12 weergeven, dan zal het programma dit getal op het scherm schrijven als:
spatie/spatie/spatie/2/2/,/1/2
Als het getal "N" cijfers voor de komma heeft, dan moeten er "M"–"N" spaties voor het getal worden opgenomen om er zeker van te zijn dat de komma op de gewenste plaats wordt geprint. In formule-vorm:

```
PRINT " "          "(1 TO M-N)
```

Vervolgens moeten we de waarde van "N" berekenen. Probeer het onderstaande programma uit met een aantal "V"-waarden tussen 1,0 en 1000000,0.

```
10 INPUT V
20 PRINT V, INT(LN V/LN 10)
30 GOTO 10
```

Als resultaat print de computer voor iedere waarde van "V" een cijfer, dat gelijk is aan het aantal cijfers voor de komma, minus 1. Een en ander is het gevolg van enige simpele wiskundige bewerkingen. Zoals men weet levert "LOG V" een getal op waarmee men 10 moet verheffen om de originele waarde "V" te krijgen. In formulevorm:
"V = 10**(LOG V)". Als we door de "INT"-functie alleen maar het gehele gedeelte van deze bewerking beschouwen, is het resultaat een cijfer dat gelijk is aan het aantal cijfers voor de komma minus 1 in het originele getal "V". Het enige probleem is, dat de ZX81 niet beschikt over de "LOG"-functie (logarithme met grondtal 10), maar over de"LN"-functie (logarithme met grondtal e). Gelukkig bestaat er een simpel verband tussen beide logarithmische uitdrukkingen, namelijk:
"LOG V = LN V / LN 10".
We kunnen dus nu de "vormgevende" regel uit het programma als volgt formuleren:

```
PRINT "          "(1 TO M-INT(LN V/LN 10));V
```

Dit lijkt al aardig op regel 3000 van het vorige programma. Alleen de termen "(V>1)" en "(V<.1)" ontbreken nog. Deze termen zijn noodzakelijk, omdat de ZX81 getallen kleiner dan 1 en groter dan 0,1 op een afwijkende manier uitprint en met deze termen kunnen we het aantal spaties aanpassen aan deze eigenaardigheid van de machine.

De "PRINT USING"-instruktie

Met de behandelde systemen voor het inkorten en afronden van getallen en het uitlijnen van de komma, kunnen we de meeste moeilijkheden bij het uitprinten van de getallen, wel de baas. Sommige BASIC-dialecten hebben een zeer werkzame instruktie, namelijk de "PRINT USING", waarmee men alle problemen rond het op de juiste manier op het scherm printen van cijferreeksen in één klap kan oplossen. We kunnen een subroutine opstellen, waarmee we bijna alle mogelijkheden van de "PRINT USING" kunnen nabootsen en die we kunnen gebruiken bij het schrijven van nieuwe pro-

gramma's of bij het verbeteren van reeds bestaande.
Door de "PRINT USING" wordt het formaat van een getal bepaald door gebruik te maken van een soort "prentje" van het getal, een prentje dat in een "string" wordt opgeslagen. Zo'n "prentje" zal er in de meeste BASIC-dialecten als volgt uitzien: " ### . ## ". Hierdoor wordt het getal opgebouwd uit drie cijfers voor en twee cijfers na de komma. Het getal 3,123 zal dan als
/spatie/spatie/3/,/1/2/ worden geprint.
Er zijn een heleboel andere symbolen die we samen met het "#"-teken kunnen gebruiken voor het vormen van het "prentje", maar een van de meest toegepaste is wel het "$"- of "£"-symbool. Als u een van beide symbolen voor het "prentje" opneemt, wordt het geldsymbool naast het meest linkse cijfer van het getal weergegeven.
Zo zal "$ ### . ### " het getal 3,1234 als /spatie/$/3/,/1/2/3/ weergeven.
Met dit "prentjes"-systeem kunnen we op een zeer eenvoudige manier zowat alle problemen op het gebied van het formeren van getallen oplossen. Als u bijvoorbeeld geen komma op het scherm wilt zien, dan past u het "prentje" aan tot " ### ".
Getallen, die te groot zijn voor de door het "prentje" geboden ruimte, worden zonder enige "format"-bewerking op het scherm geschreven.

Een uitgebreide "PRINT USING"-subroutine voor de ZX81 is zeer lang, vandaar dat we een enigszins vereenvoudigde versie gaan opstellen, waarmee we het "prentje" kunnen opbouwen door middel van de symbolen "#", "." en "$". Wél moeten we het gebruikelijke "#"-symbool vervangen door een "*", omdat de ZX81 niet beschikt over het "#"-teken.
De "PRINT USING"-subroutine gaat vergezeld van een klein testprogramma en luidt als volgt.

```
10 LET U$="$******.**"
20 PRINT TAB 9;U$
30 INPUT V
40 GOSUB 2000
50 LET V=V*100*RND
60 PRINT
70 GOTO 20
```

```
2000 LET H$=STR$ INT V
2010 LET L$=(STR$(V)) (LEN H$+1 TO)
2020 IF L$<>"" THEN IF L$(1)="."THEN LET L$
  =L$(2 TO)
2030 LET S$=U$(1)
2040 IF U$(1)<>"F" AND U$(1)<>"$"THEN LET
  S$=""
2050 LET F=0
2060 LET M=0
2070 LET N=0
2080 FOR I=1 TO LEN U$
2090 IF U$(I)="."THEN LET F=1
2100 IF U$(I)="$"AND F=0 THEN LET M=M+1
2110 IF U$(I)="$"AND F=1 THEN LET N=N+1
2120 NEXT I
2130 IF M=0 AND H$="0" THEN LET H$=""
2140 LET H$=S$+H$
2150 IF LEN H$>M THEN GOTO 2170
2160 LET H$="       "(1 TO M-LEN H$)+H$
2170 IF F<>0 THEN LET H$=H$+"."
2180 LET L$=L$+"00000000000000"
2190 IF N<>0 THEN LET H$=H$+L$(1 TO N)
2200 PRINT H$
2210 RETURN
```

Het formaat van het "prentje" wordt in de "U$"-string opgeslagen in regel 10. Door middel van de regels 20-70 wordt aan "V" een aantal testgetallen toegekend, die nadien in subroutine 2000 worden bewerkt. Eerst wordt echter het getal in "V" omgezet in een "string" door "STR$" en in twee delen gesplitst. De cijfers voor de komma gaan naar "H$" (regel 2000) en de cijfers na de komma gaan naar "L$" (regels 2010-2020).
Noteer het gebruik van de "IF ... THEN, IF ... THEN" konstruktie in regel 2020! Dit heeft dezelfde resultaten als "IF ... AND ... THEN", maar is noodzakelijk omdat in dit specifieke geval de tweede voorwaarde (L$(1)="."") alleen uitgewerkt kan worden als de eerste voorwaarde (L$ <> "") opgaat. De uitdrukking "IF L$ <> "" AND L$(1) THEN ..." zal een foutmelding opleveren als "L$" gelijk is aan nul, want dan bestaat "L$(1)" immers niet!
De variabele "S$" wordt gebruikt voor het opslaan van de glijdende geldsymbolen in de format-string "U$".
Als er geen geldsymbool wordt gebruikt, zal "S$" nul worden (regels 2030-2040). In de re-

gels 2050-2120 wordt het aantal cijfers voor de komma (M) en het aantal cijfers achter de komma (N) geteld in de format-string "U$". De variabele "S" wordt nul als er geen komma wordt teruggevonden. In regel 2020 wordt de nul voor de komma verwijderd als het getal kleiner is dan 1 of als er in het formaat van het "prentje" geen rekening mee is gehouden.
Regel 2030 is verantwoordelijk voor het opnemen van het geldsymbool vóór het getal, indien aanwezig. De spaties, die het getal aanvullen tot de afmetingen die in het "prentje" zijn vastgelegd, worden op ongeveer dezelfde manier aan het getal toegevoegd als beschreven bij het uitlijnen van de komma. Dit gebeurt in het laatste deel van het programma door de regels 2150 en 2160. Als een komma wordt verlangd, wordt deze door regel 2170 in het geheel opgenomen. Nadien wordt het gedeelte na de komma aangevuld met nullen (regel 2180 en vervolgens afgekapt op het gewenste aantal posities door regel 2190.
Tot slot zorgt regel 2200 voor het op het scherm printen van het in de goede vorm gebrachte getal.

Drie voorbeeldjes van hoe men een cijferreeks met deze subroutine kan formeren.

```
.****
.0001
.0086
.7543
30.9278
2318.9732
54566.0130
3579173.1000
62866000.0000
```

```
*.****
0.0001
0.0086
0.0058
0.2472
23.0431
2067.3304
59288.7480
3023603.1000
```

```
£********.**
          £0.01
          £0.43
         £15.15
        £993.24
      £96808.67
    £9170571.30
```

Het berekenen van rente
Als voorbeeld van hoe we de "format"-subroutine in de praktijk kunnen gebruiken, volgt nu een programma, waarmee we een tabel op het scherm kunnen printen, die een overzicht geeft van de jaarlijkse opbrengst als we iedere maand een vast bedrag tegen een vaste rente op een spaarrekening zetten.
Dit programma geeft niet alleen een fraai voorbeeld van het formeren van cijfertabellen, maar we kunnen het zowaar in de praktijk gebruiken voor het vergelijken van verschillende spaarsystemen.

Om de zaak niet al te gekompliceerd te maken, wordt er echter wel een vereenvoudigde renteberekening toegepast. Als u iedere maand een bedrag van "A" gulden spaart, dan is het jaarresultaat gelijk aan "A*12" gulden. Stel het rentepercentage gelijk aan "R" en de som van gespaard geld en rente per jaar gelijk aan "T". Het gebruikte systeem gaat er dan vanuit, dat u per jaar een bedrag van "T*R/100" aan rente op uw rekening bijgeschreven krijgt.
In dit programma maken we gebruik van de eerder gegeven subroutine 2000, die u dus in het programma moet opnemen!

```
10 PRINT TAB 8;"R E N T E   B E R E K E N I N G "
20 PRINT
30 PRINT "HOEVEEL WILT U PER MAAND"
40 PRINT "SPAREN ?";
50 INPUT A
60 PRINT A
70 PRINT "HOEVEEL JAREN ?";
80 INPUT Y
90 PRINT Y
100 PRINT "RENTE PER JAAR "
110 PRINT "IN PROCENTEN ?";
120 INPUT R
130 PRINT R
140 LET B=12*A
150 PRINT
160 PRINT "JAARLIJKS GESPAARD BEDRAG=";B
170 LET T=0
180 FOR K=1 TO Y
190 LET T=T+B
200 LET E=T*R/100
210 LET T=T+E
220 PRINT AT 20,0;"GESPAARD NA ";
230 LET U$="##"
```

```
240 LET V=K
250 GOSUB 2000
260 PRINT "JAAR";TAB 20;
270 LET U$="F########.##"
280 LET V=T
290 GOSUB 2000
300 SCROLL
310 NEXT K
320 STOP
```

Met regel 10 tot en met 110 wint de computer de nodige informatie in. Nadat het maandelijks gespaarde bedrag is omgezet in een jaartotaal (regels 140 - 160) berekent het programma het totaal gespaarde bedrag per jaar en print dit uit. Regel 190 telt deze uitkomst op bij het totaal, regel 200 berekent de nieuwe rente en regel 210 telt de rente bij het vorige totaal op. Deze berekeningen worden voor ieder jaar herhaald, tot "K" gelijk wordt aan "Y" (regel 180). De "vormgevings"-subroutine wordt gebruikt voor het opstellen van twee cijferreeksen. Enerzijds het aantal jaren, door middel van de regels 230-250, anderzijds het totaal bedrag in de regels 270-290.
Het resultaat ziet er dan ook zeer overzichtelijk uit. Vergelijk dit eens met het beeld dat ontstaat als niet van de 2000 subroutine was gebruik gemaakt!
Als u dit programma over een periode van 30 tot 40 jaar laat lopen, zult u verbaasd zijn over het eindkapitaal bij een maandelijks spaarbedrag van slechts 50 gulden!

```
        R E N T E  B E R E K E N I N G

HOEVEEL WILT U PER MAAND
SPAREN ?50
HOEVEEL JAREN ?20
RENTE PER JAAR
IN PROCENTEN ?10

JAARLIJKS GESPAARD BEDRAG =600
GESPAARD NA  1 JAAR    F660.00
GESPAARD NA  2 JAAR   F1386.00
GESPAARD NA  3 JAAR   F2184.60
GESPAARD NA  4 JAAR   F3063.06
GESPAARD NA  5 JAAR   F4029.36
GESPAARD NA  6 JAAR   F5092.30
GESPAARD NA  7 JAAR   F6261.53
GESPAARD NA  8 JAAR   F7547.68
GESPAARD NA  9 JAAR   F8962.45
GESPAARD NA 10 JAAR  F10518.70
GESPAARD NA 11 JAAR  F12230.57
GESPAARD NA 12 JAAR  F14113.62
GESPAARD NA 13 JAAR  F16184.99
GESPAARD NA 14 JAAR  F18463.48
GESPAARD NA 15 JAAR  F20969.83
GESPAARD NA 16 JAAR  F23726.82
GESPAARD NA 17 JAAR  F26759.50
GESPAARD NA 18 JAAR  F30095.45
GESPAARD NA 19 JAAR  F33764.00
GESPAARD NA 20 JAAR  F37801.50
```

8. DE ZX-PRINTER

De ZX-printer is een werkelijk ongelooflijk stuk gereedschap! Niet alleen kunnen we alles wat op het scherm staat zonder meer op papier kopiëren en programma's en gegevensbestanden uitprinten, maar met dit stukje "hard-ware" kunnen we bovendien de afmetingen en resolutie van het scherm vergroten!
Het uitbreiden van uw ZX81 komputer-systeem met een ZX-printer opent dus een geheel nieuwe reeks toepassingen!

Hoe werkt de printer

Alvorens we ons gaan verdiepen in deze nieuwe mogelijkheden willen we eerst iets vertellen over de manier waarop de printer werkt. De ZX-printer verdampt een dunne laag aluminum, die is aangebracht op een ondergrond van zwart papier. Daar waar dat aluminium verdampt, komt het zwarte papier te voorschijn en verkrijgt men een beeld, dat zonder gebruik te maken van inkt, toch als zwart op de aluminiumkleurige ondergrond verschijnt.
U kunt dit kontroleren door een stukje papier met een mesje te bekrassen. Overal waar u de aluminium laag wegkrast, komt het zwarte papier voor de dag. De printer verdampt het aluminium door gebruik te maken van elektrische vonken. Als u de printer in een verduisterde ruimte laat werken, kunt u de lichtblauwe gloed van de vonkenregen onder het staafje, waarlangs het papier wordt afgescheurd. zien.
De vonken worden geproduceerd door een aantal scherpe metalen pennen, die op een door een motor aangedreven riem zijn gemonteerd. Deze riem en de papiertransportrol worden gelijktijdig aangedreven en het resultaat is dat de metalen punten in horizontale lijnen over het papier bewegen. Tijdens dit aftasten van het papier kan een vrij hoge, maar veilige elektrische puls aan een van de naalden worden aangelegd, waardoor een vonkje ontstaat, dat een zwart puntje in het papier brandt. Als we dus een letter op het papier willen schrijven, moeten we een aantal pulsen in de juiste volgorde en op de juiste tijden naar de juiste naalden sturen, waardoor het juiste patroon van zwarte puntjes op het papier verschijnt.

Dank zij dit systeem is het niet alleen mogelijk letters weer te geven, maar letterlijk alle mogelijke symbolen en vormen. We kunnen dus op de printer zowel grafieken met hoge resolutie weergeven als de volledige set kleine letters en dat enkel met BASIC-instrukties!

Het printen met lage resolutie

Het door de ZX81 gebruikte BASIC-dialect kent drie instrukties voor de printer: "LLIST", "LPRINT" en "COPY".
De "LLIST"-instruktie kan op precies dezelfde manier worden gebruikt als de gewone "LIST"-instruktie, uiteraard met dit verschil dat in het ene geval het resultaat op papier verschijnt en in het andere geval op het scherm. De twee meest interessante instrukties zijn "LPRINT" en "COPY".
"LPRINT" wordt op dezelfde manier gebruikt als "PRINT". U kunt ook hier met "TAB"- en "AT"-instrukties de plaats bepalen waarop iets wordt geschreven. Er is echter één verschil: het papier doorloopt de printer slechts in één richting. Hij kan dus niet worden teruggedraaid! Zodoende wordt een verwijzing naar een bepaalde regelpositie in een "AT"-instruktie niet opgevolgd. Als u bijvoorbeeld:

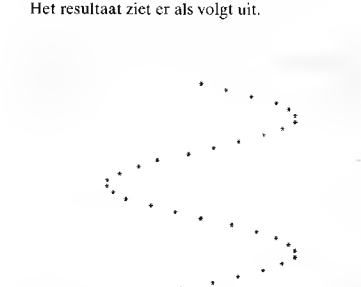
```
10 LPRINT AT 20,10;"X"
```

intoetst, zal de printer een "X"-symbool drukken op de tiende positie van de papierlijn, die op dat ogenblik in gebruik is. We kunnen dus wel de horizontale positie van de te drukken tekst onder kontrole houden, maar de vertikale positie wordt alleen maar bepaald door het in één richting lopende papiertransport.
Een en ander heeft wel als vervelende konsekwentie dat we niet klaar zijn met alle "PRINT"-instrukties in een programma te veranderen in "LPRINT" om er zeker van te zijn dat dit programma door de printer wordt gedrukt. Gelukkig kunnen we dit lastige probleem oplossen door gebruik te maken van de "COPY"-instruktie; waarover later meer.
Bij het gebruik van "LPRINT"-bevelen moet u zich voorstellen dat u op de onderste regel van

het scherm werkt en na iedere regel een "SCROLL" geeft. De printer werkt immers volgens een vergelijkbaar systeem. Het apparaat drukt een regel af, en het papiertransport kan opgevat worden als een automatische "SCROLL"! Het gevolg is dat programma's die een schermbeeld opbouwen door eerst een regel te printen en nadien te scrollen, zonder problemen via de printer op papier kunnen worden afgedrukt, door alle "PRINT""s te vervangen door "LPRINT"-instrukties en de "SCROLL"-regels te verwijderen. Op deze manier kunnen we de cijferreeksen van het statistische en het renteprogramma eenvoudig op papier zetten.
Het volgende programma, dat een sinusvormig verlopende kurve op het scherm zet,

```
10 FOR X=0 TO 8*PI STEP .3
20 LET Y=(SIN(X)+1)*15
30 PRINT AT 21,Y;"*"
40 SCROLL
50 NEXT X
```

kan zonder meer worden aangepast aan de printer, door regel 40 te schrappen en de "PRINT" te vervangen door een "LPRINT". Het resultaat ziet er als volgt uit.



Een "LPRINT" kan alle symbolen van deze ZX81 symbolenset op papier zetten en gedraagt zich precies zoals een "PRINT". Toch ontstaan er problemen als we getallen tussen 0,00001 en 0,009999999 willen uit schrijven. De eerste nul wordt als "echte" nul geschreven, dus als "0", maar de volgende als de letter O! Deze onhebbelijkheid van de machine kunnen we omzeilen door in een "LPRINT" alle getallen als "strings" op te nemen; dus door gebruik te maken van de "STRS"-funktie. Gebruik dus in plaats van:

```
10 LPRINT A
```

liever:

```
10 LPRINT STR$(A)
```

en dit soort schoonheidsfoutjes zullen zich niet meer voordoen.
Hoewel de "LPRINT"-instruktie dus erg handig is, zeker als we programma's willen laten uit schrijven om ze gemakkelijk te kunnen korrigeren, is de gemakkelijkste en meest belangrijke instruktie voor de printer de "COPY"-instruktie. Met een "COPY" wordt de volledige scherminformatie overgedragen op het papier, waarbij het niet uitmaakt wat er allemaal op het scherm staat. U kunt gebruik blijven maken van de instrukties "PRINT AT" of "PLOT/UNPLOT" om het beeldscherm op te bouwen en alvorens verder te gaan met het volgende programmadeel de beeldinhoud in zijn geheel overdragen op de papierstrook. U kunt dus als het ware programma's opstellen zonder aan het bestaan van de printer te denken en nadien met de "COPY" een exacte kopie van het beeld vereeuwigen!
We kunnen de "COPY"-instruktie op twee verschillende manier gebruiken. Als u tijdens het uitwerken van een programma de beeldinformatie wilt overdragen op papier, kunt u het programma stoppen door een "BREAK" en een "COPY"-bevel geven. Nadat de printer de beeldinhoud heeft overgenomen, kan het programma door middel van de "CONTINUE" worden voortgezet. U moet er daarbij wel voor zorgen per ongeluk geen andere instruktie te geven, want daardoor wordt het vrij moeilijk het programma weer aan de praat te krijgen en bovendien zal het scherm hierdoor worden gewist en dan is een "COPY" zinloos!
Vandaar dat de tweede manier om een "COPY" te gebruiken veel beter is, namelijk het standaard in het programma opnemen voor "COPY"-regels op de daarvoor aangewezen plaatsen. Als voorbeeld het volgende programma.

```
10 FOR I=1 TO 20
20 PRINT RND
30 NEXT I
40 COPY
50 CLS
60 GOTO 10
```

De regels 10 tot en met 30 printen 20 willekeurige getallen op het scherm, die nadien door de "COPY" in regel 40 automatisch worden overgenomen op de papierstrook. Het scherm wordt, eveneens automatisch, gewist en de volgende 20 getallen verschijnen op het scherm en daarna op papier.

Met dit idee in ons achterhoofd, kunnen we bestaande programma's zo gaan aanpassen dat de printer-mogelijkheden een integraal onderdeel van het geheel gaan vormen.
Eerst bepaalt u die plaatsen waar het zinvol kan zijn de scherminhoud over te dragen op papier, dan voegt u op die punten de vraag "wilt u een schermkopie?" in en als de gebruiker met "ja" antwoordt, wordt een "COPY"-regel geaktiveerd.
Op deze manier krijgt u steeds uitgeschreven kopieën zonder de "LPRINT"-schoonheidsfout en bovendien is deze methode de enige manier om grafieken die met behulp van de "PLOT/UNPLOT"-instrukties zijn opgebouwd op papier te krijgen.

Het printen met hoge resolutie

Door het toepassen van enige simpele subroutines kunnen we met de printer grafieken met hoge resolutie produceren, opgebouwd uit 256 bij 256 punten en alle mogelijke door de gebruiker zelf samen te stellen symbolen, de zogenoemde "user defined"-symbolen (afgekort tot "UD"-symbolen). De basis-ideeën in deze paragraaf zijn afgeleid van de demonstratieprogramma's in de handleiding die bij de printer wordt geleverd. Bovendien kunnen we deze subroutines toepassen voor het maken van eigen programma's.
Alvorens we de printer kunnen gebruiken in de hoge resolutie-modus moeten we een gedeelte van de machinecode, die is opgeslagen in het ROM-geheugen, gaan wijzigen. Hoewel het programmeren in machinecode eerst in hoofdstuk 10 wordt behandeld, kunnen we er nu toch wel mee uit de voeten als we weten dat de "LPRINT"-definitie uit het ROM-geheugen naar gelijk welk deel van de RAM-zone kan worden overgebracht en nadien gewijzigd.
Natuurlijk moeten we dan wel eerst een deel van dat RAM-geheugen reserveren voor dit doel, zodat we er zeker van zijn dat onze BASIC-instrukties nooit in dat deel van het geheugen worden opgeslagen. Dat kan door het wijzigen van het adres dat ligt opgeslagen in "RAMTOP".

```
POKE 16389,124
NEW
```

De resultaten van het printen met hoge resolutie kunnen we bewonderen aan de hand van het onderstaande programma dat een groot aantal willekeurige punten print. Het overschakelen naar de "FAST"-mode is aan te raden!

```
  10 GOSUB 1000
  20 GOSUB 3000
  30 GOSUB 2000
  40 GOTO 20

1000 IF PEEK 16388+256*PEEK 16389=31744
  THEN GOTO 1030
1010 PRINT "GEHEUGEN NIET GERESERVEERD"
1020 STOP
1030 FOR I=0 TO 112
1040 POKE 31744+I,PEEK (2161+I)
1050 NEXT I
1060 POKE 31800,63
1070 POKE 31857,201
1080 RETURN

2000 FOR H=0 TO 31
2010 POKE 16444+H,H
2020 NEXT H
2030 LET H=USR 31744
2040 RETURN

3000 FOR I=1 TO 32*8
3010 POKE 32255+I,255*RND
3020 NEXT I
3030 RETURN
```

Het resultaat:

De machinecode, die de werking van de "LPRINT" vastlegt, wordt door middel van subroutine 1000 naar de gereserveerde RAM-zone overgebracht. Met de regels 1000-1020 wordt de „RAMTOP" gewijzigd en het RAM-geheugen gereserveerd.
Nadien zal de "FOR"-lus (regels 1030-1050) 113 machinecodebytes overbrengen van 2161 (ROM-gebied) naar 31744 (gereserveerde RAM). Regel 1060 en 1070 "POKE"-en de twee noodzakelijke wijzigingen in de machinecode.
Door deze twee wijzigingen worden de 256 geheugenlokaties volgend op byte 32256 uitgeprint.
Subroutine 2000 is verantwoordelijk voor het opstarten van het printproces. De regels 2000 tot 2020 vormen een symbolenteller in het printergeheugen, startend op lokatie 16444. Regel 2030 luidt het printen in door een beroep te doen op de machinecode die door subroutine 1000 is gevormd. Dat gebeurt door:

```
USR"ADRES"
```

Hiermee kunnen we de machine naar een machinecode-subroutine sturen op een manier die te vergelijken is met de wijze waarop de "GOSUB"-instruktie de machine naar een BASIC-subroutine stuurt. De enige komplicatie is dat "USR" een functie is – net zoals "SIN" of "COS" – die dus alleen maar als uitdrukking kan worden gebruikt. Dat is de enige reden waarom regel 2030 begint met "LET H=". We zijn immers niet geïnteresseerd in de waarde die in "H" wordt opgeslagen. Ons enig streven is de machine naar de in lokatie 31744 opgeslagen machinecode te leiden!
Subroutine 3000 verandert de inhoud van het geheugendeel dat door subroutine 2000 wordt uitgeprint door willekeurige getallen te "POKE"-en. Weet u het nog?
Eén geheugenlokatie kan slechts getallen tussen 0 en 255 opslaan.
De werking van het programma moet nu duidelijk zijn.
Eerst wordt door regel 10 subroutine 1000 opgeroepen om de machinecode om te vormen. Nadien haalt regel 20 subroutine 3000 tevoorschijn, die de geheugeninhoud op een willekeurige manier verandert. Tot slot komt subroutine 2000 opdraven op bevel van regel 30 om de gegevens uit het geheugen te printen. Dit proces herhaalt zich steeds weer, tot u er genoeg van krijgt en de "BREAK" bedient.
De drie genoemde stappen, namelijk het veranderen van de machinecode, het opbouwen van een deel van het geheugen met de uit te printen informatie en het eigenlijke printen, zijn de fundamenten van het werken met grafieken met hoge resolutie. De subroutines 1000 en 2000 worden in alle in dit hoofdstuk beschreven programma's in ongewijzigde vorm gebruikt. Wel eist ieder programma uiteraard een specifieke geheugeninhoud om te kunnen worden uitgeprint.

Het tekenen van een sinuskurve

In het vorige programma hebben we de twee basis-subroutines 1000 en 2000 geïntroduceerd. Erg nuttig was het programma verder niet! Alvorens we iets zinnigs met deze techniek kunnen doen, moeten we eerst begrijpen hoe de informatie die we met subroutine 3000 in het geheugen "POKE"-en zich verhoudt tot het uiteindelijke resultaat op het papier. Helaas is dit nu net het moeilijkste punt van het werken met grafieken met hoge resolutie! Iedere keer dat subroutine 2000 in aktie komt, wordt het equivalent van een volledige rij symbolen uitgeprint. Ieder symhool wordt opgebouwd uit een matrix van 8 bij 8 punten. Een volledige regel symholen is bijgevolg samengesteld uit 32 × 8 × 8 = 2048 punten. Hetgeen heel wat meer is dan de 256 geheugen lokaties, die door subroutine 1000 worden gedefinieerd en door subroutine 2000 geprint!

Dit verschil valt te verklaren uit het feit dat iedere geheugenlokatie in staat is de toestand (dat wil zeggen ofwel zwart, ofwel wit) van 8 afzonderlijke punten te kontroleren. Iedere geheugenplaats bepaalt dus de toestand van een rij die uit 8 punten is samengesteld.
Het meest voor de hand liggend zou zijn als de eerste 32 geheugenlokaties de toestand van de eerste rij van de informatie zouden bepalen. Dat is echter niet het geval. De eerste 8 geheugenplaatsen bevatten de informatie van het eerste symbool.
De volgende 8 geheugenplaatsen bepalen de samenstelling van het tweede symbool en zo verder.
Overzichtelijk samengevat:

		eerste symbool	tweede symbool
	rij 1	geheugenplaats 1	geheugenplaats 9
	rij 2	geheugenplaats 2	geheugenplaats 10
	rij 3	geheugenplaats 3	geheugenplaats 11
regel	rij 4	geheugenplaats 4	geheugenplaats 12
	rij 5	geheugenplaats 5	geheugenplaats 13
	rij 6	geheugenplaats 6	geheugenplaats 14
	rij 7	geheugenplaats 7	geheugenplaats 15
	rij 8	geheugenplaats 8	geheugenplaats 16

Hiermee zijn we echter nog lang niet aan het eind van de moeilijke toestanden! De manier waarop iedere geheugenplaats de status van een rij uit een symbool opslaat is ook niet mis!
Moet een bepaalde punt uit zo'n rij als zwarte punt op het papier worden gedrukt, dan moet u de volgende codes "POKE"-en:

punt uit de rij	1	2	3	4	5	6	7	8
machinecode	128	64	32	16	8	4	2	1

Enige voorbeeldjes:
- als u de punt in de linker bovenhoek van het eerste symbool zwart wil maken, moet u 128 in de eerste geheugenlokatie "POKE"-en;
- als u de derde punt uit de tweede rij van het tweede symbool zwart wil laten printen, moet u 32 "POKE"-en in de tiende geheugenplaats.

Als u meer dan één punt zwart wil maken, moet u de som van de bij deze punten horende codes gaan "POKE"-en.
Als u bijvoorbeeld de vijfde en tweede punt uit de eerste rij van het eerste symbool zwart wil maken, moet u 64 + 8 = 72 in de eerste geheugenplaats "POKE"-en!

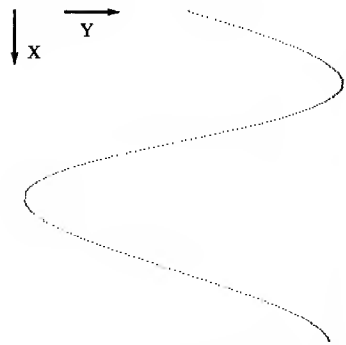
Dit alles lijkt zeer ingewikkeld en nauwelijks bruikbaar in de praktijk. Gelukkig kunnen we door middel van enige simpele BASIC-regels steeds terugvinden welke geheugenlokaties er met wat voor code ge"POKE"-ed moeten worden om een bepaald puntenpatroon op te bouwen.
We weten nog dat een symbool bestaat uit 8 × 8 punten, namelijk 8 rijen van 8 punten horizontaal en 8 kolommen van 8 punten vertikaal. Als u zich de 32 symboolposities van een regel voorstelt als opgebouwd uit 32 × 8 = 256 punten per rij, genummerd van links naar rechts (kolomvolgnummer) en uit 8 kolommen, genummerd van boven naar beneden (rijvolgnummer), dan kan een willekeurige punt met rijvolgnummer "X" (horizontaal) en kolomnummer "Y" (vertikaal) als een zwarte punt worden weergegeven.

```
100 LET C=INT(X/8)
110 LET B=7-X+C*8
120 POKE 32255+C*8+Y,2**B
```

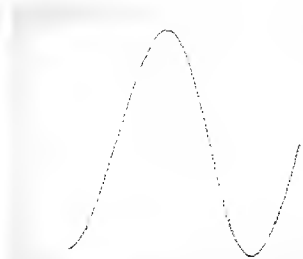
Regel 100 berekent de symboolpositie en regel 110 bepaalt de puntpositie in de symboolplaats. We hebben nu voldoende kennis vergaard om een sinusvorm (of gelijk welke andere wiskundige uitdrukking) op het papier uit te tekenen. Ook dit programma kan beter in de "FAST"-mode worden uitgedraaid!

```
 10 LET L=0
 20 GOSUB 1000
 30 FOR X=0 TO 8*PI STEP .03
 40 LET Y=(SIN(X)+1)*127
 50 GOSUB 3000
 60 NEXT X
 70 STOP

3000 LET Y=INT Y
3010 LET K=INT(Y/8)
3020 LET R=7-Y+K*8
3030 FOR I=0 TO 31
3040 POKE 32256+I*8+L,0
3050 IF K=I THEN POKE 32256+I*8+L,2**R
3060 NEXT I
3070 LET L=L+1
3080 IF L=8 THEN GOSUB 2000
3090 IF L=8 THEN LET L=0
3100 RETURN
```

Wat het volgende beeld oplevert.



De eerste regel (10) zet de rijen-teller "L" op nul. Als "L" gelijk wordt aan 8 hebben we 8 rijen punten vastgelegd en wordt het tijd subroutine 2000 op te roepen om dit deel van de grafiek uit te tekenen (regel 3070). Regel 20 bepaalt de machinecode-subroutine. Met de regels 20-50 wordt de sinusfunctie uitgerekend voor waarden tussen 0 en 8*PI. De kurve wordt uitgeprint terwijl het papiertransport loopt, dus de sinuswaarde "Y" wordt opgebouwd in de looprichting van het papier.
De schaalwaarde van "X" doet niet ter zake omdat iedere nieuwe berekende waarde van "Y" op de volgende regel wordt geschreven!
"Y" mag niet zo groot worden dat de kurve gedeeltelijk buiten het papier valt, vandaar dat "Y" zo wordt geschaald dat de laagste sinuswaarde 0 is en de hoogste 254.
Ook nu gebruiken we subroutine 3000 voor het wijzigen van de geheugeninhoud, zodat het gewenste patroon van zwarte punten ontstaat.
Voor iedere berekende sinuswaarde "Y" wordt de symboolplaats en puntpositie bepaald. De volledige rij wordt gelijk gemaakt aan nul (regels 30303-3060), waardoor de vorige printinformatie verdwijnt en het volgende punt op de juiste plaats wordt ge"POKE"ed (regel 3050).
Met regel 3070 wordt de inhoud van de rijenteller met 1 verhoogd.
Met deze basistechniek is het mogelijk grafieken van zeer hoge kwaliteit op het papier te drukken. U kunt nu diverse wiskundige uitdrukkingen laten uittekenen en proberen grafieken te verfraaien door het aanbrengen van een assenstelsel. Wel moeten we toegeven dat deze techniek verre van gemakkelijk is en bovendien tamelijk traag werkt!

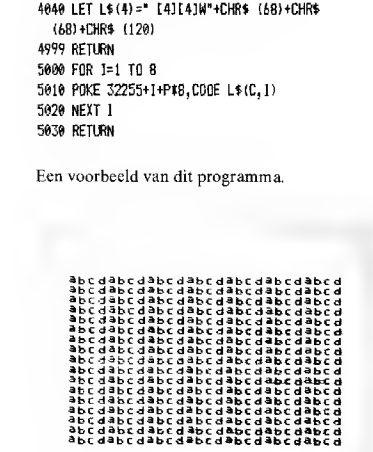


Een kleine letter- of speciale symbolenset
Met de behandelde technieken kunnen we de beschikbare symbolenset belangrijk uitbreiden. Een van de meest nuttige toepassingen is de uitbreiding met een kleine letterset.
Het onderstaande programma vormt de basis van een tekstprocessor, die de beschikking heeft over zowel kleine letters als hoofdletters.

```
 10 GOSUB 1000
 20 GOSUB 4000
 30 LET A$="ABCD"
 40 GOSUB 3000
 50 FOR P=1 TO LEN A$
 60 LET C=CODE A$(P)-37
 70 GOSUB 5000
 80 NEXT P
 90 GOSUB 2000
100 STOP

1000 IF PEEK 16388+256*PEEK 16389=31744
  THEN GOTO 1030
1010 PRINT "GEHEUGEN NIET GERESERVEERD"
1020 STOP
1030 FOR I=0 TO 112
1040 POKE 31744+I,PEEK (2161+I)
1050 NEXT I
1060 POKE 31800,63
```

```
1070 POKE 31857,201
1080 RETURN
2000 FOR H=0 TO 31
2010 POKE 16444+H,H
2020 NEXT H
2030 LET H=USR 31744
2040 LET L=0
2050 RETURN
3000 FOR I=0 TO 31*8
3010 PPOKE 32255+1,0
3020 NEXT I
3030 RETURN
4000 DIM L$(26,8)
4010 LET L$(1)=" S[4]W"+CHR$ (68)+"W "
4020 LET L$(2)=" RNORNO"+CHR$ (120)+CHR$
  (68)+CHR$(68)+CHR$ (120)+" "
4030 LET L$(3)=" SRNORNORNDS "
4040 LET L$(4)=" [4][4]W"+CHR$ (68)+CHR$
  (68)+CHR$ (120)
4999 RETURN
5000 FOR I=1 TO 8
5010 POKE 32255+I+P*8,CDOE L$(C,I)
5020 NEXT I
5030 RETURN
```

Een voorbeeld van dit programma.



De subroutines 1000 en 2000 worden weer op de bekende manier gebruikt voor het opbouwen van de machinecodesubroutine en voor het uitschrijven van de inhoud van het geheugendeel. Subroutine 4000 bepaalt het puntenpatroon van ieder nieuw symbool dat u wilt bezigen. Het patroon voor het "I"-de symbool wordt opgeslagen in "L$(I)" als een "string" van acht symbolen.

Als u een nieuw puntenpatroon wilt vastleggen, moet u de vorm van het symbool uittekenen in een matrix van 8 bij 8 punten. Behandel rij voor rij en noteer de code van ieder zwart punt. Tel nadien alle codes bij elkaar op. Nadat u op deze manier alle 8 rijen onder handen hebt genomen, ontstaat een lijstje van 8 getallen tussen 0 en 255. Ieder getal vormt de code van het symbool dat in de "string" moet worden opgeslagen om het gewenste patroon van witte en zwarte punten te verkrijgen. Als we deze cijferreeks in de vorm van een "string" willen opslaan, is het noodzakelijk ieder getal om te vormen tot het korresponderende symbool door gebruik te maken van de "CHR$"-functie.

Om plaats (en intoets-tijd) te besparen, is het aan te bevelen deze symbolen rechtstreeks vanaf het toetsenbord in te voeren. U kunt dus beter een spatie intoetsen dan de uitdrukking "CHR$(0)"!

De gang van zaken kan het best worden toegelicht aan de hand van een voorbeeld. Stel dat we de kleine letter a willen invoeren. Het puntenpatroon in de 8 bij 8 matrix en de korresponderende codes zijn:

```
.  .  .  .  .  .  .  .                              0
.  .  .  .  .  .  .  .                              0
.  .  *  *  *  .  .  .     32 + 16 + 8  =           56
.  .  .  .  .  *  .  .                              4
.  .  *  *  *  *  .  .  32 + 16 + 8  + 4  =         60
.  *  .  .  .  *  .  .              64 + 4  =        68
.  .  *  *  *  *  .  .  32 + 16 +  8 + 4  =         60
.  .  .  .  .  .  .  .                              0
```

Deze codes worden nadien omgezet in de volgende symbolen, door gebruik te maken van de string "L$(1)":

```
L$(1)=" S[4]W"+CHR$(68)+"W"
```

Merk op dat, als er geen symbool is dat korrespondeert met de symboolcode er niets anders opzit dan gebruik te maken van de "CHR$"-functie. Als u een sleutelwoord zoals "LET" in de "string" moet gebruiken, kunt u ofwel de "CHR$" gebruiken, ofwel het sleutelwoord rechtstreeks invoeren door "THEN" in te toetsen (voor de "K"-kursor), nadien "LET" en tot slot de edit-toetsen gebruiken voor het wissen van de "THEN".

Nadat het puntenpatroon is opgebouwd in "L$" kunt u een reeks symbolen opslaan in "A$" (regel 30), waarna de regels 40-90 de nieuwe vormen in een regel op de printer zullen uitschrijven.
De manier waarop de standaard symbolenset korrespondeert met de nieuwe vormen wordt bepaald door regel 60.
In dit geval komt "A" overeen met het patroon dat ligt opgeslagen in "L$(1)". Regel 70 roept subroutine 5000 op, die het puntenpatroon in "L$(C)" opslaat in de symboollokatie "P". Nadat alle puntenpatronen zijn behandeld zal een beroep op subroutine 2000 ervoor zorgen dat de geheugeninhoud wordt uitgeschreven.

Besluit

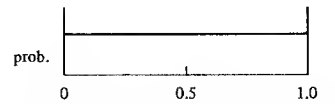
Denk nu niet dat alle toepassingen van de kombinatie ZX81 en ZX-printer in dit hoofdstuk aan de orde zijn geweest! Er kan veel meer mee worden uitgevoerd!
Zo zou u het programma voor de kleine letter symbolenset kunnen omzetten in een echte tekstprocessor met zowel kleine letters als hoofdletters. De tekst zou dan eerst op het scherm kunnen worden gezet met geïnverteerde symbolen op de plaats van de hoofdletters en normale op de plaats van de kleine letters. Nadat het scherm vol staat, zou het programma alle symbolen omvormen tot respektievelijk hoofdletters en kleine letters, zodat de printer een stuk geschreven tekst aflevert, dat er uitziet zoals we dat gewend zijn.
Een tweede projekt is het opstellen van een grafisch pakketje voor de ZX81, waarmee we wat gemakkelijker met technieken met hoge resolutie kunnen werken.
De mogelijkheden zijn eindeloos, het initiatief ligt in uw handen!

9. STATISTISCHE TECHNIEKEN VOOR GEVORDERDEN

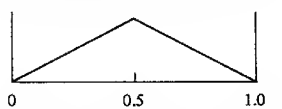
Willekeur is een fundamenteel gegeven bij het programmeren van spelletjes op een computer. De meeste boeken of handleidingen, die spelletjes-programma's beschrijven, gaan dan ook in het kort in op de mogelijkheden van het opwekken van willekeurige getallen. We veronderstellen dat de lezer van dit boek, ook al een enigszins gevorderde programmeur, (anders had hij niet de behoefte gehad de geheugenkapaciteit van zijn machine uit te breiden) niet onkundig is van de basis-technieken.
Toch bestrijken deze basis-technieken maar een heel klein deel van alles wat gedaan kan worden met willekeurige getallen. Het werken met statistische programma's is een van de belangrijkste toepassingsgebieden van het serieuze computerwerk!
In dit hoofdstuk behandelen we eerst de basistechnieken voor het opwekken van willekeurige getallen en vervolgens wat minder voor de hand liggende toepassingen van het werken met willekeur, namelijk enige nuttige statistische programma's.

Statistische verdelingen

Stel dat we in een programma waarmee we weersverwachtingen willen simuleren, willekeurige temperaturen moeten opwekken. We kunnen dit bijvoorbeeld doen op de manier die in de meeste boeken wordt beschreven: het totale gebied van aanvaardbare temperaturen in een aantal delen verdelen, ieder deel koppelen aan een hepaald getal en dan de computer getallen willekeurig laten berekenen. Een nadere studie van de "RND"-functie leert ons dat dit een tamelijk omslachtige methode is, want met de "RND"-functie kunnen we getallen opwekken, die groter of gelijk zijn aan nul, maar kleiner dan 1.
De volgorde van deze getallen is niet te voorspellen en volstrekt willekeurig. In principe liggen er in het interval tussen 0 en 1 een oncindig aantal verschillende getallen, maar omdat de computer bepaalde afrondingen toepast, is dit aantal in de praktijk eindig.
Met de "RND"-functie kunnen we dus aan een continue gebeurtenis een willekeurig verlopende waarde toekennen.
Het volgende merkwaardige feit is, dat de "RND"-functie slechts één van de vele mogelijkheden is om een willekeurige cijferreeks op te wekken.
Iedere "kansgenerator" zal op zijn specifieke wijze de willekeurige spreiding van de in het toegestane interval opgewekte getallen uitvoeren. De manier waarop de kans op het opwekken van een bepaald getal varieert kan grafisch worden voorgesteld in een zogenoemde kansdistributiegrafiek. Zo weten we dat de "RND"-functie getallen in het interval 0 tot 1 opwekt en dat ieder getal even veel kansen heeft.
De kansdistributiegrafiek voor de "RND"-functie ziet er dus als volgt uit.



Om voor de hand liggende redenen wordt deze distributie de "uniforme distributie" genoemd.
Een andere "kansgenerator" zou een distributie kunnen geven, die er zo uitziet.



Bij deze kansgenerator neemt de kans op het verkrijgen van een hepaald getal toe als dit getal dichter in de buurt van de waarde 0,5 komt. Dit soort kansverdeling is zeer nuttig voor het simuleren van verschijnselen, waarbij een gemiddelde waarde waarschijnlijker is dan een extreme. Neem bijvoorbeeld het spelletje, waarbij het de bedoeling is de verkoop van frisdrank te raden in de maand juli.
De verkoop van frisdrank is deels afhankelijk van de temperatuur. We weten uit ervaring dat

het in de maand juli tamelijk warm kan zijn. Maar we weten ook dat er in juli koude en zeer warme dagen kunnen voorkomen. Driehoeksdistributie is dan een ideale manier om de temperatuur zo realistisch mogelijk in het spel te verwerken. U moet dan de volgende uitdrukking gebruiken.

2*M*TRI

waarbij "M" staat voor de gemiddelde temperatuur in juli en "TRI" de nieuwe functie is, die getallen volgens de driehoeksdistributie opwekt. Dat we het resultaat nog eens met twee moeten vermenigvuldigen wordt veroorzaakt door de keuze van het totale interval van de distributie tussen 0 en 1. De meeste getallen zullen dus rond de 0,5 liggen en het is logisch dat de temperatuur zonder deze exta vermenigvuldigingsfactor de helft zou zijn van de voor juli geldende normale temperaturen.

Het probleem is echter dat er geen BASIC-functie "TRI" ter beschikking staat!

We moeten dus een subroutine gaan opstellen die ervoor zorgt dat getallen volgens de gewenste distributiekurve worden gegenereerd.

Voor de driehoeksdistributie gaat dat vrij eenvoudig. Als we twee getallen, die volgens de uniforme distributie van de "RND"-functie zijn gegenereerd bij elkaar optellen, ontstaat het gewenste resultaat.

De formule voor de driehoeksdistributie luidt dus:

LET T = (RND + RND)/2

Uit de waarschijnlijkheidsrekening zijn een aantal zeer nuttige distributies bekend, die echter niet zo gemakkelijk op onze computer zijn na te bootsen als de driehoeksverdeling. De volgende tabel geeft de voornaamste distributies en hun typische toepassingen.

Uit deze soorten distributie een gemotiveerde keuze te doen, kan alleen na grondige analyse van het te programmeren probleem.

Een uitgewerkt voorbeeld van de toepassing van de poisson-distributie volgt later in dit hoofdstuk. In de eerste plaats gaan we ons bezig houden met het opstellen van de subroutines nodig voor het nabootsen van de vijf genoemde distributies.

De werking van deze subroutines te doorgronden zonder kennis van de basisbeginselen van de waarschijnlijkheidsrekening is erg moeilijk. Maar ook zonder deze kennis kunnen we ze wel toepassen!

soort distributie	typische toepassingen
normale	voor het simuleren van "natuurlijke" processen met meetfouten
chi-kwadraat	voor statistische berekeningen bij het nemen van monsters
exponentieel	voor economische berekeningen, zoals het simuleren van wachttijden of tijdsduur tussen twee cliënten
binomiaal	voor het simuleren van de kans dat een bepaald onderdeel uit een reeks niet aan de specificaties voldoet
poisson	voor het simuleren van het aantal te verwachten cliënten in een bepaald tijdsinterval

Als u een inzicht wilt krijgen in de vorm van de distributies, kunt u de subroutine voor het opwekken van willekeurige getallen in het statistische programma vervangen door een van de volgende en telkens een aantal histogrammen laten uitprinten. Vergeet niet dat aan de parameters geëigende waarden moeten worden toegekend.

De normale distributie

Omdat dit een van de belangrijkste statistische functies is, geven we twee subroutines. De eerste is gebaseerd op het centrale limiet theorema en geeft een benadering, die voor de meeste toepassingen nauwkeurig genoeg is.

```
1000 LET Z=0
1010 FOR I=1 TO N
1020 LET Z=Z+RND
1030 NEXT I
1040 LET Z=SQR(3/N)*(2*Z-N)
1050 RETURN
```

De nauwkeurigheid van deze formule neemt toe bij een stijgende waarde van "N". Bruikbare waarden voor "N" liggen tussen 20 en 50. De tweede subroutine is gebaseerd op de methode die door Box-Muller is ontwikkeld en wordt alleen toegepast als er hoge eisen aan de nauwkeurigheid worden gesteld.

```
1000 LET Z=SQR(-2*LN(RND))*COS(PI*RND)
1010 RETURN
```

Voor beide subroutines geldt dat "Z" een willekeurig getal met een normale distributie is, met een gemiddelde waarde van nul en een standaardafwijking van 1.

De chi-kwadraat distributie
Als u een distributie met twee vrijheidsgraden nodig heeft, kunt u hiervan gebruikmaken.

```
1000 LET X=-2*LOG(RND)
```

Meer algemeen, voor een distributie met een even aantal vrijheidsgraden (D = 2N), geldt:

```
1000 LET U=1
1010 FOR I=1 TO D
1020 LET U=U*RND
1030 NEXT I
1040 LET X=-2*LOG(U)
```

Om een oneven aantal vrijheidsgraden op te wekken, kunnen we gebruik maken van:

```
1000 LET Y=X+Z*Z
```

waarbij geldt dat "X" een willekeurig getal is met een chi-kwadraat distributie met 2N vrijheidsgraden en Z normaal met een gemiddelde waarde van 0 en een standaardafwijking van 1.

De exponentiële distributie
Hiervoor staat de volgende subroutine ter beschikking.

```
1000 LET X=-(1/L)*LN(RND)
1010 RETURN
```

waarbij "X" voldoet aan de distributie:
1 – EXP (–L*X).

De binomiale distributie
In deze subroutine heeft "X" een binomiale distributie, staat "N" voor het totaal aantal gegenereerde getallen en geeft "P" de kans op sukses.

```
1000 LET X=0
1010 FOR I=1 TO N
1020 IF RND>P THEN GOTO 1040
1030 LET X=X+1
1040 NEXT I
1050 RETURN
```

De poisson-distributie
Deze subroutine wordt gebruikt in het simulatieprogramma, dat we verderop in dit hoofdstuk zullen behandelen.

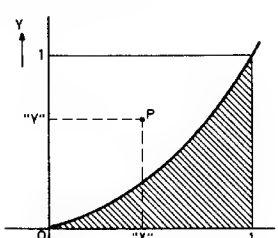
```
1000 LET X=0
1010 LET E=EXP -M
1020 LET P=1
1030 LETP=P*RND
1040 IF P<E THEN RETURN
1050 LET X=X+1
1060 GOTO 1030
```

Hierbij staat "X" voor een geheel getal tussen 0 en oneindig, met als distributie M**X*EXP(–M)/X!.

Monte Carlo integratie en de berekening van de waarde van PI
Tot nu toe hebben we nog geen nuttige toepassing van het gebruik van willekeurige getallen gegeven.
Er zijn toevalsgetallen nodig om voorziene en onvoorziene gebeurtenissen te kunnen simuleren. Een voorbeeld van zo'n simulatie vindt u aan het einde van dit hoofdstuk.
We kunnen met toevalsgetallen een reeks ingewikkelde wiskundige vergelijkingen oplossen.
Deze methode staat bekend onder de naam "monte carlo", en zoals we zullen zien, niet voor niets genoemd naar het beroemde casino.
Stel dat we de oppervlakte, die wordt omsloten door een curve bepaald door een wiskundige vergelijking, moeten herekenen. De meest gebruikelijke manier om zo'n probleem op te lossen, is het toepassen van wiskundige fomules. Soms is het toch nog moeilijk om met klassicke wiskundige theorieën tot een oplossing te komen en moeten we een geheel andere benaderingswijze uitvoeren. De integratie-theorieën gaan uit van een flink brok moeilijke wiskunde, maar het gebruiken van het gezond verstand leidt soms tot zeer verrassende oplossingen. Zo ook hier!

We nemen de uitdrukking:
$Y = X^2$
en moeten deze integreren voor waarden van "X" tussen 0 en 1. We moeten dus de oppervlakte berekenen, die wordt ingesloten door de kurve $Y = X^2$; de horizontale as van het assen-

stelsel ("Y" = 0) en de lijn die wordt gegeven door de functie "X" = 1.



We zouden in dit simpele geval de bekende simpson-regel van de integraalrekening kunnen toepassen, maar stel dat we die niet kennen en dat we trouwens geen enkel idee hebben van hoe we zo'n probleem wiskundig moeten aanpakken. Laten we dan even ons gezonde verstand gebruiken!

Als we twee willekeurige getallen beschouwen, "X" en "Y", tussen de waarden 0 en 1, dan zullen beide getallen een punt "P" bepalen, dat ergens in het vierkant ligt en dat wordt begrensd door de waarden "X" = 1 en "Y" = 1. Dat punt "P" kan dan ofwel boven, ofwel onder de curve liggen. In het getekende geval zijn "X" en "Y" per toeval zo gekozen, dat het punt "P" boven de grafiek ligt.

Het zal duidelijk zijn dat de kans dat het punt "P" onder de curve ligt, wordt bepaald door de totale oppervlakte onder de curve! Dit is het geheim van de monte-carlo-integratie! Als we een hele reeks van paren van willekeurige getallen opwekken en we berekenen steeds of het punt "P", dat door een paar wordt bepaald, onder of boven de curve ligt, dan zal de verhouding tussen het aantal punten onder de curve en het totaal aantal punten een maat zijn voor de oppervlakte onder de grafiek.

Uiteraard geeft deze verhouding slechts een benadering van de gezochte oppervlakte, maar die benadering wordt wel steeds nauwkeuriger als we het aantal paren laten toenemen.

Passen we deze techniek eens toe om de oppervlakte onder de "X-kwadraat"-kurve te berekenen met als grenzen "X" = 1 en "Y" = 0.

Het simpele programma ziet er als volgt uit.

```
10 LET H=0
20 LET N=0
30 LET X=RND
40 LET Y=RND
50 IF Y<X*X THEN LET H=H+1
60 LET N=N+1
70 PRINT AT 21,0;"OPPERVLAK=";H/N,"N=";N
80 SCROLL
90 GOTO 30
```

Met de variabele "H" tellen we het aantal punten onder de curve en met "N" het totaal aantal punten.

Met de regels 30 en 40 genereren we twee willekeurige getallen. De berekening van regel 50 onderzoekt of het punt boven of onder de curve ligt. Als het onder de curve ligt, wordt de waarde van variabele "H" met één eenheid verhoogd. Door middel van regel 70 wordt de benadering van de oppervlakte en het aantal punten-paren op het scherm geprint.

Het resultaat:

```
OPPERVLAK=1             N=1
OPPERVLAK=1             N=2
OPPERVLAK=0.66666667 N=3
OPPERVLAK=0.5           N=4
OPPERVLAK=0.4           N=5
OPPERVLAK=0.33333333 N=6
OPPERVLAK=0.42857143 N=7
OPPERVLAK=0.375         N=8
OPPERVLAK=0.33333333 N=9
OPPERVLAK=0.3           N=10
OPPERVLAK=0.27272727 N=11
OPPERVLAK=0.25          N=12
OPPERVLAK=0.23076923 N=13
OPPERVLAK=0.21428571 N=14
OPPERVLAK=0.2           N=15
OPPERVLAK=0.1875        N=16
OPPERVLAK=0.17647059 N=17
OPPERVLAK=0.16666667 N=18
OPPERVLAK=0.21052632 N=19
```

Als u dit programma een hele tijd laat doorlopen, zult u vaststellen dat de benadering van de oppervlakte, die in het begin sterk varieert van berekening tot berekening, zeer langzaam naar een bepaalde vaste waarde gaat. Uit de traditionele formules weten we dat de exacte waarde van de oppervlakte gelijk is aan 1/3 of 0,33333.

Waarschijnlijk zal het u zwaar tegenvallen hoe lang het duurt voor de monte-carlo-methode een enigszins nauwkeurig antwoord heeft uitgewerkt. Zelfs na 100 herekeningen is de benadering nog maar 0.352 en het tweede cijfer na de komma varieert nog steeds van berekening tot berekening!
Het lijkt er dus op, dat deze monte-carlo-integratie niets meer dan een leuk spelletje is, met weinig praktisch nut. Dat klopt, als we ons beperken tot zeer eenvoudige problemen zoals het hier behandelde. Als de vraagstelling echter veel ingewikkelder wordt, zoals het drie-dimensionaal werken en het berekenen van het volume dat onder een golvend oppervlak ligt, is de monte-carlo-integratie een van de beste en snelste methoden die ter beschikking staan!
Als tweede voorheeld van de mónte-carlo-integratie geven we een programma, waarmee we de waarde van PI kunnen berekenen.
Uit de traditionele wiskunde weten we dat het integreren van de curve:
1/(X*X + 1)
tussen 0 en 1 als resultaat PI/4 oplevert.
Als we dus PI willen herekenen, moeten we het resultaat van de met monte-carlo gevonden benadering met vier vermenigvuldigen.
De volgende twee regels moeten worden aangepast.

```
50 IF Y<1/(X*X+1) THEN LET H=H+1
70 PRINT AT 21,0;"PI=";4*H/N,"N=";N
                  ⋮
```

Laat het programma lopen en ga er even rustig bij zitten.

```
PI=3.1894934        N=533
PI=3.1910112        N=534
PI=3.1850467        N=535
PI=3.1865672        N=536
PI=3.1880819        N=537
PI=3.1821561        N=538
PI=3.1836735        N=539
PI=3.1851852        N=540
PI=3.1792976        N=541
PI=3.1734317        N=542
PI=3.174954         N=543
PI=3.1764706        N=544
PI=3.1779817        N=545
PI=3.1794872        N=546
PI=3.1809872        N=547
PI=3.1824818        N=548
PI=3.1839709        N=549
PI=3.1854546        N=550
PI=3.1869329        N=551
PI=3.1884058        N=552
PI=3.1898734        N=553
PI=3.1913357        N=554
                  ⋮
```

Wie heeft voldoende geduld om te waehten tot de computer in de buurt van PI = 3,14159 komt?

Willekeurige getallen en hun problemen

De betrouwbaarheid van de resultaten van de meeste nuttige toepassingen van het werken met willekeurige getallen wordt bepaald door de "kwaliteit" van deze getallen. Dit probleem doet zich niet voor als we spelletjes met willekeurige getallen ontwerpen, we verwachten dan niet meer dan dat een speler de volgorde van de gegenereerde getallen als volkomen willekeurig ervaart. Bij technieken als de monte-carlo-integratie worden veel hogere eisen gesteld aan de kwaliteit van de generator, die de willekeurige getallen opwekt. Als deze generator bijvoorbeeld vaker getallen in de buurt van de 1 opwekt dan in de buurt van de 0, dan zal het schatten van de oppervlakte onder een curve duidelijk worden beïnvloed door deze afwijking van de generator.
In feite zouden we dus technieken moeten ontwikkelen, waarmee we de kwaliteit van een generator van willekeurige getallen kunnen beoordelen. Dat is allesbehalve eenvoudig! Dit houdt in dat deze testprogramma's de generator moeten laten lopen en de recks getallen die hij opwekt — door middel van statistische berekeningen — op hun "mate van willekeur" moeten kunnen beoordelen.
Zo zouden we kunnen onderzoeken of alle getallen even vaak voorkomen, door een histogram te plotten, waarin een flink aantal getallen zijn ondergebracht.
Toch bestaat er een zeer eenvoudige methode om de kwaliteit van een kansgenerator te verbeteren.
Het onderstaande geeft de typische procedure weer.

1 — vul een "array" "A" met afmeting "N" met willekeurige getallen van de verdachte generator;

2 – wek een willekeurig geheel getal "R" op ergens tussen 1 en "N" en verruil de inhoud van "A(1)" met "A(R)";
3 – herhaal deze procedure voor ieder element van het "array";
4 – gebruik de "N" willekeurige getallen in het "array" als de volgende "N" in de reeks.

Met de volgende subroutine kunnen we deze procedure in een willekeurig programma inbouwen.

```
10 DIM A(N)

5000 FOR I=1 TO N
5010 LET A(I)=RND
5020 NEXT I
5030 FOR I=1 TO N
5040 LET T=A(I)
5050 LET R=INT(RND*N)+11
5060 LET A(I)=A(R)
5070 LET A(R)=T
5080 NEXT I
5090 RETURN
```

Als algemene regel geldt dat de kwaliteit van het resultaat verbetert als men het "schudden" langer volhoudt! Helaas vergt deze procedure heel wat tijd en als we een heleboel willekeurige cijfers nodig hebben kunnen we beter het probleem aan de basis aanpakken: de kwaliteit van de generator verbeteren!
Toch kunnen we met deze "schud"-techniek de kwaliteit van een slechte generator aanmerkelijk verbeteren! Gebruik dus deze subroutine in geval van twijfel.

Een zakelijk simulatieprogramma

Als voorbeeld van het niet wiskundig gebruik van de beschreven technieken, gaan we het volgende praktijkvoorbeeld behandelen.
Een warme bakker wil op voorhand bepalen hoeveel broden hij moet bakken om met zo weinig mogelijk oud brood te blijven zitten en zo min mogelijk klanten te moeten teleurstellen.
Als we er van uitgaan dat de cliëntèle iedere dag volgens hetzelfde patroon in- en uitloopt, kunnen we de poisson-distributie gebruiken om de dagelijkse toeloop te simuleren. Deze distributie is immers bij uitstek geschikt voor het simuleren van gebeurtenissen die per tijdseenheid steeds dezelfde kans op "het zich voordoen" hebben. Als een bepaald soort ongeval op ieder moment kan gebeuren, dan zal het totaal aantal van dat soort ongevallen over een bepaalde tijd worden gegeven door de poissondistributie.
Terug naar onze warme bakker! We zullen het gemiddelde aantal klanten per dag moeten weten. We kunnen dat aantal bepalen door het aantal klanten over een bepaald aantal dagen te tellen (inclusief degenen die teleurgesteld zonder brood naar huis moeten worden gestuurd) en dit aantal te delen door het aantal dagen van de telperiode. Stellen we het gemiddelde aantal klanten per dag gelijk aan "M", dan zal volgens een poissondistributie het echte aantal per dag rond het gemiddelde "M" schommelen.
De waarschijnlijkheid om op een bepaalde dag "X" kopers te mogen verwelkomen, wordt gegeven door:

$$\frac{M^X \, EXP(-M)}{X!}$$

Om de gevolgen voor onze klanten vast te stellen als we per dag slechts "S" broden hakken, moeten we getallen gaan genereren volgens de poissondistributie met als gemiddelde waarde "M" en gaan berekenen hoe vaak het volgens de poissondistributie berekende aantal groter of kleiner is dan "S".
In de vorm van een programma.

```
 10 LET L=0
 20 LET W=0
 30 PRINT TAB 8;"S E R V I C E "
 40 PRINT AT 5,0
 50 PRINT "GEMIDDELD AANTAL KLANTEN"
 60 PRINT "PER DAG ";
 70 INPUT M
 80 PRINT M
 90 PRINT "HOEVEEL KLANTEN KUNT U"
100 PRINT "PER DAG HELPEN ?";
110 INPUT S
120 PRINT S
130 FOR I=1 TO 30
140 GOSUB 1000
150 IF X>S THEN LET L=L+X-S
160 IF X<S THEN LET W=W+S-X
170 PRINT AT 21,0;"DAG ";I;"KLANTEN= ";X
180 SCROLL
190 NEXT I
200 PRINT
```

```
210 SCROLL
220 PRINT "VERLOREN KLANDIZIE=";L
230 SCROLL
240 PRINT "AFGESCHREVEN PRODUCTEN=";W
250 STOP

1000 LET X=0
1010 LET E=EXP-M
1020 LET P=1
1030 LET P=P*RND
1040 IF P<E THEN RETURN
1050 LET X=X+1
1060 GOTO 1030
```

Regel 30 tot en met 120 vragen de nodige gegevens voor het berekenen van de diverse grootheden. Het gemiddelde aantal kopers wordt opgeslagen in "M" en het aantal klanten dat per dag kan worden geholpen (dus het aantal per dag te bakken broden, in de veronderstelling dat iedere klant slechts één brood afneemt) wordt ondergebracht in "S". Met de regels 130 tot 190 simuleren we het aantal klanten dat iedere dag de bakkerij zal bezoeken. Voor iedere dag in een periode van 30 dagen wordt door middel van subroutine 1000, die we herkennen als een poissondistributie, de waarde van "X" berekend. Dit getal wordt vergeleken met het aantal gebakken broden. Als er meer kopers binnenkomen dan er broden voorradig zijn, wordt het verschil berekend in regel 150. Als er daarentegen minder bezoekers zijn dan er broden in het rek liggen, wordt het overschot aan onverkocht brood berekend in regel 160. Na de dertigdaagse onderzoeksperiode wordt een som gemaakt van het aantal weggestuurde klanten en van het totaal aantal onverkochte broden en deze twee gegevens worden uitgeprint.
Met dit simulatieprogramma kan de bakker bepalen of hij meer of minder broden moet gaan bakken. Natuurlijk zijn er een helebocl factoren, die deze uiterst eenvoudige simulatie kunnen verstoren. Als bijvoorbeeld de winst op de verkoop van een brood veel groter is dan de kostprijs zou het toch de moeite waard kunnen zijn veel meer broden te bakken dan noodzakelijk en het verlies aan grondstoffen op de koop toe te nemen. U kunt dit programma dus behoorlijk gaan verfijnen, door bijvoorbeeld in te kalkuleren wat het kost om een klant te moeten afwijzen en wat het kost om een brood te moeten weggooien. U zou dan de aangepaste winst kunnen berekenen en deze voor verschillende hoeveelheden brood kunnen laten uitprinten.
Men kan dit programma gebruiken als basis voor het onderzoeken van de gang van zaken voor iedere winkel, waar de voorraden vastliggen, zoals bijvoorbeeld krantenkiosken.
Het resultaat van dit programma.

```
        S E R V I C E

GEMIDDELDE AANTAL KLANTEN
PER DAG 50

HOEVEEL KLANTEN KUNT U
PER DAG HELPEN ? 60

DAG 1 KLANTEN=46
DAG 2 KLANTEN=57
DAG 3 KLANTEN=41
DAG 4 KLANTEN=50
DAG 5 KLANTEN=56
DAG 6 KLANTEN=48
DAG 7 KLANTEN=44
DAG 8 KLANTEN=53
DAG 9 KLANTEN=54
DAG 10 KLANTEN=52
DAG 11 KLANTEN=49
DAG 12 KLANTEN=48
DAG 13 KLANTEN=60
DAG 14 KLANTEN=51
DAG 15 KLANTEN=61
DAG 16 KLANTEN=42
DAG 17 KLANTEN=57
DAG 18 KLANTEN=45
DAG 19 KLANTEN=49
DAG 20 KLANTEN=57
DAG 21 KLANTEN=52
DAG 22 KLANTEN=52
DAG 23 KLANTEN=44
DAG 24 KLANTEN=51
DAG 25 KLANTEN=39
DAG 26 KLANTEN=45
DAG 27 KLANTEN=37
DAG 28 KLANTEN=41
DAG 29 KLANTEN=46
DAG 30 KLANTEN=53

VERLOREN KLANDIZIE=1
AFGESCHREVEN PRODUKTEN=330
```

We hebben dit programma met opzet eenvoudig gehouden, zodat het erg duidelijk is hoe alles in zijn werk gaat. Het ligt voor de hand dat

dit soort zakelijke simulaties in de praktijk veel ingewikkelder zijn, omdat we ook rekening moeten houden met het gedrag van mensen. Zo zou het best eens kunnen gebeuren, dat een vaste klant van deze bakker voorgoed naar de konkurrent gaat als hij een keer nee verkocht krijgt! Dit soort effecten is uiteraard moeilijk in berekeningen te verwerken.

Sinclair's driehoek

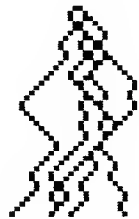
Tot slot van dit "ernstige" hoofdstuk een luchtige noot. Sinclair's driehoek is een voorbeeld van een willekeurige beweging, dat we voor ons plezier kunnen programmeren. Een voorwerp voldoet aan een willekeurige beweging, als zijn verplaatsing op de een of andere manier door de wetten van de waarschijnlijkheid wordt beheerst. In de natuurkunde vinden we talloze voorbeelden van willekeurige bewegingen. De meest bekende is de "Brownbeweging", genoemd naar de ontdekker, van vloeistofdeeltjes in een vloeistof als gevolg van botsingen van de moleculen. Ook stuifmeelkorrels in de lucht voeren willekeurige bewegingen uit, veroorzaakt door de wind.

We kunnen zo'n beweging simuleren door een punt te printen met coördinaten ("X", "Y") en dit punt nadien te verplaatsen door bij beide coördinaten een bepaald willekeurig getal op te tellen.

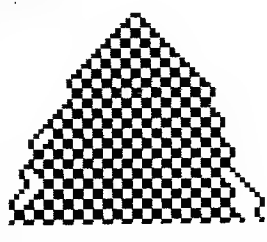
Het onderstaande programma van Sinclair's driehoek geeft een simulatie van de weg, die een bliksemflits door de lucht aflegt.

```
 10 LET X=15
 20 LET I=0
 30 FOR Y=1 TO 21
 40 IF RND>.5 THEN GOTO 80
 50 LET I=I+1
 60 PRINT AT Y,X+I;"[T]"
 70 GOTO 100
 80 PRINT AT Y,X+I;"[Y]"
 90 LET I=I-1
100 NEXT Y
110 GOTO 10
```

Met regel 10 bepalen we het punt van waaruit de bliksemflits ontstaat. De regels 20 tot en met 100 plotten punten volgens een willekeurige lijn van boven naar beneden over het scherm. Door middel van regel 40 bepalen we of de volgende positie naar links of naar rechts moet. In het eerste geval zorgen regel 80 en 90 voor het uitplotten van de nieuwe positie, in het tweede geval wordt deze taak overgenomen door de regels 50 en 60. Als u dit programma laat "RUN"-nen, zult u een aantal onregelmatige zig-zag-lijnen over het scherm zien ontstaan.



Laat u de machine echter een hele tijd lopen, dan zal er langzaam maar zeker een schaakhord-vormig patroon ontstaan. Dit patroon kan niet anders dan driehoekvormig zijn, omdat alle "bliksemflitsen" uit een en hetzelfde punt ontspruiten.



Dit speelse programma kan bijvoorbeeld worden gebruikt om de laserstraal uit een ruimtepistool te simuleren!

Besluit

In dit hoofdstuk hebben wij bij lange na niet alle toepassingen van willekeurige getallen behandeld. We wilden u alleen maar een richtlijn geven om zelf te bepalen of simulatietechnieken nuttig kunnen zijn in uw eigen specifieke toepassingen. Heeft deze verhandeling een gevoelige snaar geraakt, dan raden wij u aan gespecialiseerde literatuur te gaan lezen.

10. HET PROGRAMMEREN IN MACHINECODE

Hoewel we met de 16K-versie van de ZX81 zeer uitgebreide programma's kunnen uitwerken, heeft de computer toch het nadeel dat zijn verwerkingssnelheid voor een heleboel zaken, zoals het tegelijkertijd laten bewegen van verschillende objekten of het testen van zijn eigen geheugen, veel te traag is.
We zullen het geduld moeten opbrengen om iets langer te wachten op het resultaat van een programma. Als we bedenken dat de ZX81 slechts een fraktie kost van de prijs die we voor een grote computer moeten betalen en toch tot dezelfde dingen in staat is, kunnen we hem zijn traagheid gemakkelijk vergeven!
Toch zijn er toepassingen waar de traagheid van de machine, zelfs in de "FAST"-mode, te groot is om er op een redelijke manier gebruik van te maken. Als het antwoord op een vraag minutenlang op zich laat wachten is de kans groot dat de gebruiker van de computer ondertussen een kopje koffie is gaan drinken en het hele programma al weer is vergeten! Verder is best een uitgebreide versie van een populair spel als "ruimte veroveraars" in BASIC te programmeren, maar het zou waarschijnlijk het traagste spel zijn dat ooit voor een computer is geschreven. Dit soort spelletjes moet het nu net van snelheid hebben.
Kortom, waarvoor u de ZX81 ook gebruikt, er komt altijd een moment waarop u wordt geconfronteerd met zijn snelheidsgrens.
Het verrassende is, dat we de oplossing niet moeten zoeken in het kopen van een duurdere en krachtigere microcomputer. Alle microcomputers en misschien zelfs alle computers, zullen voor sommige toepassingen te traag blijken te zijn. We kunnen de snelheid van onze computer echter zo verhogen dat hij alles kan wat een grote machine kan. Maar daar staat wel tegenover, dat we de zo vertrouwde BASIC-taal zo snel mogelijk moeten vergeten en moeten gaan wennen aan iets geheel nieuws: de **Z80-MACHINECODE!**
Laat u door niemand iets op de mouw spelden, machinecode is veel moeilijker dan BASIC! Waarom zou men anders die trage BASIC nog gebruiken?

Bovendien moeten we ons behoorlijk diep in de materie van machinecode-programmering storten alvorens we tot iets behoorlijks in staat zijn.
We zijn ons er terdege van bewust, dat we bij u nu al een behoorlijke afkeer van machinecode aan het kweken zijn, maar we hopen toch dat de rest van dit hoofdstuk u voldoende aanleiding zal geven om over uw angst heen te stappen en de wondere wereld van het programmeren in machinecode onbevangen te betreden.
Wel moet u zich goed realiseren dat machinecode niet op één zonnige namiddag kan worden geleerd. Maar probeer door te zetten, het is de moeite waard!

De traagheid van BASIC

BASIC behoort tot de zogenoemde hogere programmeertalen, die ter beschikking staan voor het programmeren van computers. De meeste computers kunnen in hogere talen, zoals BASIC, FORTRAN, ALGOL en dergelijke worden geprogrammeerd. Zelfs de kleine ZX81 zou in principe in gelijk welke hogere taal met zijn gebruiker kunnen kommuniceren. Maar daar BASIC aan de meeste wensen tegemoet komt, heeft nog niemand de moeite genomen de machine aan te passen aan een andere taal.
Computers kunnen verschillende talen verstaan, omdat iedere hogere taal wordt omgezet in een veel fundamentelere taal alvorens het programma wordt uitgevoerd. Deze fundamentele taal noemt men machinecode en dit is de enige taal die rechtstreeks door de machine kan worden verwerkt. Het probleem is dat iedere machine zijn eigen machinecode heeft en dat dus iedere machine de hogere talen op een specifieke manier moet vertalen.
De konsekwentie is dus dat er niet zoiets bestaat als **DE** machinecode! De code die een machine verstaat wordt bepaald door de microprocessor die in het apparaat is toegepast. Zo'n microprocessor is een zeer ingewikkeld elektronisch onderdeel en vormt het hart van de computer. De ZX81 past een Z80-microprocessor toe en bijgevolg moeten we ons de Z80-machinecode eigen maken. In feite is het helemaal geen slechte keuze voor de eerste kennismaking met een ma-

chinecode, want de Z80 wordt in heel wat computers toegepast.
We moeten ons wel realiseren dat een aantal populaire microcomputers zoals de APPLE en de PET niet rond de Z80 zijn opgebouwd.
De ZX81 moet de door u ingetoetste BASIC-instrukties eerst in Z80-machinecode vertalen, alvorens hij er iets mee kan doen.
Dat vertaalproces kan op twee manieren worden aangepakt. Er zijn machines die het volledige BASIC-programma in één keer omzetten in machinecode en dan aan de slag gaan; zo'n vertaalprogramma noemt men een "compiler" (letterlijk vertaald: verzamelaar). Dit soort machines is meestal niet zo gemakkelijk te programmeren als de ZX81; die gebruikt een andere techniek.
Bij de Sinclair computer wordt iedere BASIC-regel eerst vertaald en dan uitgewerkt. Het BASIC-sleutelwoord wordt in de machine vergeleken met een vertaallijst. Wordt het sleutelwoord in deze lijst gevonden, dan weet de machine wat voor soort akties er van hem worden verlangd.
Als de BASIC-instruktie bijvoorbeeld "GOTO 10" luidt, dan zondert de machine het sleutelwoord "GOTO" af uit de regel en zoekt in de ingebouwde lijst op wat er bij een "GOTO" te gebeuren staat. In dit specifieke geval vinden we in deze tabel het machinecode-ekwivalent voor: "werk de uitdrukking achter 'GOTO' uit, zoek in het programma de regel met hetzelfde regelnummer en zorg dat deze instruktie wordt uitgevoerd."
De omzetting van BASIC in machinecode noemt men "interpreting" en het programma "interpreter", letterlijk vertaald: tolk. Omdat de ZX81 iedere BASIC-regel eerst gaat "interpreteren", werkt het geheel erg traag. Alvorens het BASIC-bevel uit te voeren, moet de machine heel wat tijd spenderen aan het opzoeken van wat deze BASIC-regel nu wel precies wil zeggen!
Bij rechtstreeks gebruik van machinecode in programma's doen dit soort vertaalproblemen zich natuurlijk niet voor. Vandaar dat zo'n programma wel tien keer sneller door de machine kan worden uitgevoerd!

De karakteristieken van machinecode

Het ligt voor de hand ons af te vragen waarom we niet vaker van machinecode gebruik maken, als deze taal zo snel is. Het antwoord op deze vraag hebben we in de inleiding van dit hoofdstuk reeds even aangestipt: programmeren in machinecode is veel ingewikkelder dan programma's opstellen in BASIC. En, vreemd genoeg, dit verschijnsel wordt nu net veroorzaakt doordat machinecode zo'n eenvoudige taal is!
BASIC is een taal, die geschreven is naar menselijke begrippen. Als u ergens de BASIC-uitdrukking:

```
LET A = B + C*2 - Z
```

ziet staan, dan weet u precies wat daarvan de bedoeling is en dat het resultaat van deze berekening in "A" wordt opgeslagen. Machinecode werkt niet met zulke "ingewikkelde" dingen als vermenigvuldigen en delen. Machinecode kent slechts twee bewerkingen: optellen en aftrekken en deze bewerkingen kunnen slechts één voor één in een bepaalde geheugenplaats worden uitgevoerd. Moeilijke bewerkingen zoals vermenigvuldigen, moeten dus opgesplitst worden in een aantal simpelere bewerkingen. Het vermenigvuldigen van twee getallen kan in machinecode alleen door herhaaldelijk optellen.
Het is zeer zeker niet de bedoeling u te onderwijzen in machinecode. Wat we wel hopen is dat u, door wat volgt, de smaak van machinecode-programmering te pakken krijgt en enig inzicht opdoet over de fundamentele ideeën. De beste manier om dit te bereiken is: enige simpele voorbeelden behandelen. Vooreerst moeten we iets meer weten over de architectuur van de in de ZX81 gebruikte microprocessor Z80.

De Z80 - microprocessor

Hoewel de Z80 een van de snelste en veelzijdigste microprocessors is die op de markt zijn, kan hij toch maar zeer eenvoudige bewerkingen uitvoeren. Hij kan zich toegang verschaffen tot iedere geheugenplaats, maar is niet in staat de inhoud van zo'n geheugenplaats rechtstreeks te wijzigen.
Ieder bewerking moet in speciale geheugenplaatsen in de microprocessor zelf worden uitgevoerd. Deze speciale geheugenplaatsen noemt men "registers" en omdat er maar een beperkt aantal beschikbaar zijn, worden ze door middel van een naam in plaats van door middel van adressen aangeduid.
Uit het totale aantal zullen we er vier bespreken.

– het A-register, dat "accumulator" wordt genoemd;
– het B-register;
– het H-register en
– het L-register.

Het A-register is het meest nuttige en wordt vooral voor wiskundige bewerkingen gebruikt. Het ziet er net zo uit als een gewone geheugenlokatie en kan getallen tussen 0 en 255 opslaan. Het B-register staat ter beschikking voor algemene toepassingen, voornamelijk voor het bewaren van tijdelijke resultaten zonder dat het externe geheugen hoeft te worden aangesproken. In dit B-register kunnen slechts een beperkt aantal bewerkingen worden opgeslagen en ook in dit register kunnen we getallen tussen 0 en 255 onderbrengen. Ook de H- en L-registers staan ter beschikking voor algemene toepassingen, maar omdat ze meestal gecombineerd worden toegepast, spreekt men over het HL-paar. Door het invoeren van een registerpaar kunnen we natuurlijk de hoeveelheid op te bergen informatie aanzienlijk uitbreiden. We hebben dan voldoende ruimte om het adres van gelijk welke geheugenlokatie uit het totale geheugen van de machine op te bergen. Het HL-paar dient voornamelijk voor dit doel; vandaar dat men dit paar het adresregister noemt. Bij een heleboel Z80-berekeningen wordt het in het HL-register opgeslagen getal gebruikt als het adres van de geheugenlokatie die aan de orde is.
Dit hele verhaal over registers en wat we ermee kunnen doen, krijgt meer inhoud als we enige voorbeelden van Z80-instrukties gaan behandelen.

De "LD"- en "ADD"-instrukties

De Z80 herkent machinecode-commando's aan hun codegetallen. Een machinecode-programma is dus niets anders dan een lange lijst met getallen die in het geheugen van de computer wordt opgeslagen. Het probleem hierbij is dat wij, mensen, niet zo goed in staat zijn te begrijpen wat deze onoverzichtelijke reeks van getallen nu wel mag voorstellen. Getallen vormen niet onze natuurlijke menselijke taal! Om het geheel wat toegankelijker te maken, gaan we daarom die codegetallen vervangen door iets meer zeggende symbolen.
Als wij bijvoorbeeld het A-register willen gaan laden met de inhoud van geheugenlokatie 123, kunnen we de volgende versie van de "LD"-instruktie gebruiken:

```
LD A, (123)
```

hetgeen u kunt lezen als: "laad het A-register met de informatie van geheugenlokatie 123". Het is namelijk in de Z80-machinecode een ongeschreven regel dat alles wat tussen haakjes staat wordt geïnterpreteerd als een adres.
Voor mensen is de instruktie duidelijk, de Z80 kan echter geen letters lezen; dus moet deze boodschap op de een of andere manier worden vertaald in cijfers.
Daarvoor staat een lijst ter beschikking, de zogenoemde instruktielijst en als u daarin opzoekt wat de code is voor de instruktie "LD A", dan vindt u het getal 58.
Deze instruktielijst vindt u in appendix A van de ZX81-handleiding. Deze cijfercode staat bekend onder de naam "operation code" of "op-code". De "op-code" voor "LD A" vanuit een geheugenlokatie is dus 58.
De kompleet gecodeerde instruktie bevat uiteraard ook het adres van de geheugenlokatie die naar het A-register moet worden overgebracht. Dit adres wordt opgenomen na de "op-code". De instruktie "LD A, (123)" wordt dus als volgt in machinecode geschreven: 58,123,0. De nul is noodzakelijk, omdat er voor het bewaren van een adres steeds twee geheugenlokaties worden gebruikt en voor adressen kleiner dan 255 moet dus dat tweede, ongebruikte, geheugenplekje worden opgevuld met een nul.
Voor het opslaan van zo'n simpele instruktie als het laden van de accumulator hebben we dus drie geheugenplaatsen nodig. Er zijn Z80-instrukties die met minder geheugenruimte toe kunnen, maar er zijn er ook die er meer dan drie nodig hebben!
Als tweede voorbeeld van een machinecode-instruktie behandelen we:

```
ADD A,n
```

Hierbij staat "n" voor een getal tussen 0 en 255. Met deze instruktie wordt het getal "n" opgeteld bij de inhoud van het A-register. Als er in de accumulator reeds een getal van 3 was opgeslagen, zal de geheugeninhoud worden verhoogd tot 8 na het uitvoeren van de instruktie "ADD A, 5".

De "op-code" voor deze instruktie is 198 en het getal "n" wordt in de eerstvolgende geheugenplaats opgeslagen. Omdat de waarde van "n" tussen 0 en 255 moet liggen, hebben we slechts één geheugenlokatie nodig. De komplete machinecode voor "ADD A,5" luidt dus "198,5" en kan in twee geheugenplaatsen worden opgeslagen.
Men schrijft een machinecode-programma door gebruik te maken van het soort uitdrukkingen als "LD" en "ADD". Nadat het volledige programma op papier staat, duikt men in de instruktielijst en gaat moeizaam alle notaties vervangen door de bijbehorende codenummers. Het omzetten van symbolen in codenummers is zo'n standaard-procedure, dat er een kompleet programma voor bestaat en men het aan de machine kan overlaten! Zo'n omzettingsprogramma gaat door het leven onder de naam "assembler" en dank zij dit soort programma's wordt het programmeren in machinecode toch iets wat nog te doen valt! Helaas heeft de ZX81 niet zo'n assembler als standaard-pakket, maar we zouden het zelf kunnen opstellen of kant-en-klaar kopen hij een van de talloze software leveranciers, die zich gespecialiseerd hebben in het ontwikkelen van software voor Sinclair computers.

Een klein programma als voorbeeld

We willen zeer zeker niet de suggestie wekken dat na deze inleiding en het behandelen van twee machinecode-instrukties u voldoende weet van het onderwerp om te beginnen met het schijven van machinecode-programma's. Wél moet u in staat zijn te hegrijpen hoe zo'n programma in principe werkt. In de "BASIC ROM" zit een kleine machinecode-subroutine (startend op adres 2056) waardoor de machine het symbool, waarvan de machinecode in de accumulator staat, op het scherm print.
Nu we dit weten, kunnen we een machinecode-programma ontwerpen, waardoor het scherm wordt volgeschreven met een door ons gekozen symbool.

```
START LD A,21
      CALL 2056
      JP START
```

Als gevolg van de eerste instruktie van dit programma, wordt de accumulator geladen met het getal 21, de machinecode voor het "+"-symbool. De tweede instruktie is het machinecode-ekwivalent van een BASIC "GOSUB" en door deze "CALL" gaat de machine als volgende stap naar adres 2056 waar, zoals we nu weten, een subroutine start die ervoor zorgt dat de inhoud van de accumulator op het scherm wordt geprint. Het resultaat van deze "CALL" is dus dat er op het scherm een plusje verschijnt. De laatste instruktie is het machinecode-ekwivalent van de BASIC "GOTO" en de machine wordt daardoor naar de "START" van het programma gestuurd. "JP" is de afkorting van "JUMP", de laatste instruktie zegt dus niets anders dan: "JUMP TO START" of "spring naar start". De werking van dit programma is dus vrij snel te doorgronden; het print het scherm vol plusjes en als het scherm vol is, verschijnt er een foutcode.
In de praktijk moeten we er dus voor zorgen dat de machinecode-subroutine wordt gestopt nadat de computer hem heeft uitgewerkt. Voor de eenvoud hebben we dat nu niet gedaan, vandaar dat het programma vanzelf stopt als het scherm vol staat en er een fout wordt gerapporteerd.
De drie regels vormen een kompleet uitgewerkte machinecode-programmering. Natuurlijk moeten we deze mededeling nu nog gaan coderen en testen.
Coderen is niets anders dan de door mensen gebruikte taal omzetten in cijfercodes, die de machine hegrijpt. Voor de eerste twee regels van het programma gaat dat zonder problemen.

instructie	*code*
START LD A,21	62,21
CALL 2056	205,8,8

De code voor "LD A", gevolgd door een getal, is "62" en het getal dat in de accumulator moet worden geladen volgt direkt op de op-code, hier 21 dus.
De "op-code" voor "CALL" is "205" en het adres waar de "CALL" naar verwijst, moet in de twee volgende geheugenlokaties worden ondergebracht. Omdat dit adres groter is dan 255, moet het volgens de bekende systemen worden opgesplitst in twee getallen, dus 2056 = 8 + 8 * 256.
Bij het omzetten van de laatste regel van het programma stuiten we op een probleem! De

"op-code" voor de "JP" is zo op te zoeken: "195". Het onderbengen van het "JUMP"-adres is ook geen punt; daarvoor staan de twee volgende geheugenlokaties ter beschikking. Waar we echter op vastlopen is dat we niet weten op welke geheugen-lokatie het programma start!
We moeten nu eerst gaan bepalen waar we dit programma in het geheugen willen onderbrengen. Het totale geheugen heeft een aantal zones, die geschikt zijn voor het onderbrengen van machinecode-programma's. In hoofdstuk 8 hebben we gezien dat machinecode kan worden opgeslagen in een gedeelte van het geheugen na "RAMTOP".

In dit hoofdstuk gaan we echter een praktischer manier voor het opbergen van machinecode toepassen. We zullen een "REM"-instruktie aan het begin van het programma opnemen en de codes daarin verwerken door simpelweg de bij iedere code behorende toets in te drukken!
De vijf eerste codes van dit programma waren "62, 21, 205, 8, 8". Deze codes komen overeen met de volgende toetssymbolen: "Y, +, LN, (A), (A)".
We kunnen nu de volldige machinecode als volgt in de "REM-instruktie opnemen.

```
10 REM Y+LN(A)(A)
```

Hierbij staat "LN" voor één enkel symbool, dat korrespondeert met de "LN"-functie van het toetsenbord en dit symbool wordt ingevoerd door het na elkaar indrukken van de functie-toets en de "Z". "(A)" staat voor het grafische symbool van de A-toets.
Nu kunnen we de laatste regel van het programma afwerken, want we kunnen het start-adres van het programma gaan berekenen. Alle BASIC-programma's starten op geheugenlokatie 16509. Er worden twee bytes gebruikt voor het opslaan van het regelnummer, evenveel voor de lengte van de regel en één voor het sleutelwoord "REM". Een simpel optelsommetje leert dat het eerste symbool van de "REM" in lokatie 16514 thuis hoort. De laatste regel van het programma wordt bijgevolgd: "JP 16514", hetgeen wordt gecodeerd als:

```
JP 16514
195,130,64
```

Ook deze codegetallen moeten natuurlijk in de "REM" worden opgenomen, maar alweer worden we geconfronteerd met een mocilijkheid! Wat is namelijk het geval? De symbolen, die overeen komen met de codes 130 en 64 zijn respektievelijk (W) en "RND", dat is geen punt.Voor het getal 195 staat echter geen symbool ter beschikking! Deze code kan dus niet rechtstreeks in de "REM"-instruktie worden opgenomen.
Wat nu?
We nemen een spatie op in de "REM"-instruktie en gaan de juiste code in die spatie opnemen door middel van een "POKE"! Met deze "POKE" gaan we dus de code voor het spatie-symbool vervangen door de code, waarvoor geen symhool ter beschikking staat.
We kunnen nu weliswaar de volledige "REM"-instruktie opnemen, aangevuld met een "POKE"-regel, maar we weten nu nog niet hoe we de machine aan het verstand moeten brengen, dat we machinecode willen invoeren.
Bij het bespreken van het gehruik van de ZX-printer in hoofdstuk 8 hebben we gezien dat we de "USR"-functie kunnen gebruiken om toegang te krijgen tot een machinecode-subroutine. Even recapituleren.

```
USR adres
```

leidt de machine naar een machinecode-subroutine, die start op de geheugenlokatie met nummer "adres".
Omdat "USR" een functie is en geen instruktie, moeten we het opnemen in een opdrachtinstruktie, zoals:

```
LET A=USR adres
```

In het kader van deze voorbeelden zijn we niet geïnteresseerd in de waarde, die in "A" wordt opgeslagen als gevolg van de "USR"-functie.
We weten ondertussen het startadres van de machinecode-subroutine (16514), zodoende wordt de uitgewerkte "USR":

```
LET A=USR 16514
```

Het is best de moeite waard een volledig overzicht te geven van alle adressen die we voor dit programma gebruiken en van wat er allemaal in die geheugenplaatsen is opgeborgen.

adres	*code*	*symbool*
16514	62	Y
16515	21	+
16516	205	LN
16517	8	[A]
16518	8	[A]
16519	195	geen
16520	130	[W]
16521	64	RND

Het nut van deze tabel is dat we nu in één oogopslag kunnen zien dat de op-code waarvoor geen symbool ter beschikking staat (195, weet u nog?) naar adres 16510 moet worden ge-' 'POKE"-ed!

We zijn nu eindelijk zo ver, dat we het kompleet uitgewerkte programma kunnen presenteren.

```
10 REM Y+LN[A][A] [W]RND
20 POKE 16519,195
30 LET A=USR 16514
```

Het is van het uiterste belang dat de "REM"-regel precies zo wordt ingevoerd als genoteerd. Dus: "LN" en "RND" door middel van één toetsdruk, "(A)" en "(W)" onder de vorm van hun grafische symbolen en één spatie tussen de tweede "(A)" en de "(W)".
U zult konstateren dat dit programma het scherm in een zeer hoog tempo vult met allemaal plusjes. U kunt de verwerkingssnelheid vergelijken met hetzelfde programma in BASIC:

```
40 PRINT "+"
50 GOTO 40
```

Een opmerkelijk verschil!
In de inleiding van dit programma hebben we gesteld dat we machinecode-programma's ook kunnen kontroleren. Dat is niet zo moeilijk, het enige wat moet gebeuren is het toevoegen van onderstaande subroutine, waardoor de inhoud van iedere geheugenlokatie in de "REM" op het scherm wordt gezet.

```
60 FOR I=16514 TO 16521
70 PRINT I,PEEK I
80 NEXT I
```

U kunt dit programma aktiveren door "GOTO 60" in te toetsen.
Met de bespreking van dit programma hebben we, zo hopen we, voldoende aangetoond dat we de moeilijkheidsgraad van het programmeren in machinecode niet hebben overschat!
Hoewel dit zeer simpele programma maar drie machinebewerkingen bevat, kost het heel wat tijd om het op te stellen en nog veel meer tijd om uit te leggen hoe alles werkt.
Toch kunt u aan de hand van dit voorbeeld een objektieve kijk krijgen op het grote voordeel van machinecode, door het vergelijken van de verwerkingssnelheden van hetzelfde programma geschreven in BASIC en geschreven in machinecode.

Een tweede voorbeeld: het inverteren van het scherm

Dit tweede voorbeeld beschrijft een machinecode-subroutine, waarmee we de inhoud van het scherm kunnen inverteren. Met andere woorden: alles wat wit was wordt zwart en vice versa. Dit is een zeer nuttig programma: we kunnen er knipperende mededelingen mee op het scherm schrijven, hetgeen goed van pas komt bij het programmeren van spelletjes of als we in een "serieuze" toepassing de aandacht op een bepaalde mededeling willen vestigen.
Daarnaast is het een goed voorbeeld van hoe lange machinecode-programma's worden opgebouwd. Het programma is echter zo uitgebreid dat we niet op alle details kunnen ingaan zoals dat bij het eerste voorbeeld is gedaan.
Wat we wél doen, is het systeem verklaren en een korte toelichting geven bij iedere instruktie. Ook de volledige codering wordt gegeven, maar niet uitvoerig besproken.
In principe komt het er op neer, dat we iedere symboolcode vermeerderen met het getal 128. Uit de gegevens in appendix A van de ZX81-handleiding, blijkt immers dat de codes van een symbool en zijn inverse steeds 128 verschillen. Het opstellen van die waarde bij een symboolcode heeft dus tot gevolg, dat het symbool in zijn inverse wordt omgezet. Minder voor de hand liggend is dat het nogmaals optellen van 128 het originele symbool te voorschijn tovert. Toch is dat niet zo moeilijk in te zien. Zoals we weten, kunnen we in de accumulator getallen tussen 0 en 255 opslaan. Als de inhoud 255 is en we tellen er 1 bij op, dan "springt" de inhoud

van het register als het ware terug naar 0 zonder dat er een foutmelding optreedt. Als u dus bij de inhoud van het A-register een getal optelt en de som van beide getallen is groter dan 255, dan zal de inhoud van het register gelijk worden aan de som minus 256.
Het twee maal achter elkaar optellen van de waarde 128 herstelt dus als het ware de originele inhoud van het register, want twee maal 128 is 256!
Een voorbeeld:
Stel, dat in het register het getal 38 staat, de code voor de letter "A". Tel daar 128 bij op, door middel van een "ADD A, 128"-instruktie. Het resultaat is 38 + 128 = 166, de code voor het geïnverteerde "A"-symbool. Geef nu weer dezelfde instruktie. In het register zou een waarde van 166 + 128 = 294 worden opgeslagen, maar dit getal is groter dan de maximale inhoud van het register en dus zal de teller na het bereiken van inhoud 255 teruggaan naar 0 en vanaf deze waarde verder tellen. Het resultaat wordt 294 – 256 = 38, de code voor het originele symbool "A"!
Samengevat: het optellen van het getal 128 bij de inhoud van het register inverteert het symbool en dezelfde bewerking voor de tweede maal uitgevoerd, herstelt het originele symbool.
Dit was even een moeilijk puntje, de rest van het programma gaat recht op het doel af!
Eerst zoeken we het adres op van de eerste geheugenlokatie van de schermbeeldzone van het geheugen. Nadien wordt dit adres in de accumulator opgeslagen en wordt er 128 bij opgeteld. Vervolgens zetten we deze nieuwe code van het geïnverteerde symbool weer op de originele plaats terug.
Wel zitten er enige addertjes onder het gras!
Op de eerste plaats moeten we rekening houden met de code van een "terug naar de kantlijn"-commando, want het zal duidelijk zijn, dat we deze code niet mogen veranderen.
Op de tweede plaats moeten we ervoor zorgen dat we na het behandelen van 21 regels het programma laten stoppen. Beide problemen hebben iets met elkaar te maken.
Telkens als we de code voor een "newline"-commando tegenkomen, weten we immers dat we één regel van het scherm onder handen hebben genomen.
Beide problemen kunnen we dus in één keer oplossen door de inhoud van het A-register te kontroleren. Als deze gelijk wordt aan 118, de code voor een "terug naar de kantlijn"-instruktie, mag de inhoud niet met 128 worden vermeerderd. Bovendien gaan we het aantal keren dat we code 118 in de accumulator tegenkomen, tellen in het B-register. Uit praktische overwegingen draaien we dit laatste om: we zetten eerst het getal 21 in het B-register en gaan die waarde telkens als we in de accumulator code 118 tegenkomen met één verminderen. Na 21 regels is de inhoud van het B-register gelijk aan nul en dit kunnen we gemakkelijk vaststellen.
Het enige, wat we nog moeten vertellen is dat een machinecode-programma steeds eindigt met een "RET"-instruktie. waardoor de machine weet dat er nadien weer naar BASIC wordt omgeschakeld.
Het komplete programma ziet er als volgt uit:

adres	*instructie*	*code*
16514	LD HL,(16396)	42,12,64
16517	LD B,21	06,21
16519	INC HL	35
16520	LD A,(HL)	126
16521	CP 118	254,118
16523	JP Z,16532	202,148,64
16526	ADD A,128	198,128
16528	LD (HL),A	119
16529	JP 16519	195,135,64
16532	DEC B	05
16533	JP NZ,16519	194,135,64
16536	RET	201

In het kort wat kommentaar:
- **LD HL,** (16396) laadt het HL-paar met het adres van het begin van de beeldzone;
- **LD B, 21** zet het getal 21 in het B-register, waarmee we het aantal regels zullen gaan tellen;
- **INC HL** verhoogt de waarde in het HL-paar met 1;
- **LD A, (HL)** transporteert het getal in het HL-paar naar de accumulator;
- **CP 118** vergelijkt de inhoud van de accumulator met 118, waardoor we te weten komen wanneer de computer aan een nieuwe regel begint;
- **JP Z, 16532** stuurt de machine naar adres 16532 als de inhoud van de accumulator gelijk is aan 118;
- **ADD A, 128** verhoogt de inhoud van de ac-

cumulator met 128 waardoor het symbool wordt geïnverteerd;
- **LD (HL), A** stuurt de inhoud van de accumulator, de code van het geïnverteerde symbool, terug naar de oorspronkelijke plaats;
- **JP 16519** stuurt de machine naar adres 16519;
- **DEC B** vermindert de waarde van register B met een eenheid;
- **JP NZ, 16519** stuurt de machine naar adres 16519, als de inhoud van B nog niet gelijk is aan nul;
- **RET** kondigt het einde van het machinecode-programma aan.

De volgende stap is het omzetten van alle instrukties in de bijbehorende machinecodes.

```
adres    code     symbool
16514    42       E
16515    12       £
16516    64       RND
16517    06       [T]
16518    21       +
16519    35       7
16520    126      geen
16521    254      RETURN
16522    118      geen
16523    202      ASN
16524    148      inverse =
16525    64       RND
16526    198      LEN
16527    128      [)
16528    119      geen
16529    195      geen
16530    135      [3]
16531    64       RND
16532    05       [5]
16533    194      TAB
16534    135      [3]
16535    64       RND
16536    201      TAN
10 REM E£RND[T]+7 RETURN ASN [=]RNDLEN
 [)[3]RND[5]TAB[3]RNDTAN
20 POKE 16520,126
30 POKE 16522,118
40 POKE 16528,119
50 POKE 16529,195
```

Hieruit blijkt dat er vier codes zijn, die we niet rechtstreeks door middel van symbolen in een "REM"-instruktie kunnen onderbrengen. We moeten dus op de bekende manier vier "POKE"-bevelen opnemen en de "REM"-regel voorzien van de nodige spaties. In hoofdstuk 8 hebben we uitgelegd hoe we sleutelwoorden als "RETURN" kunnen invoeren.
Het volledige programma luidt:

```
 60 PRINT "$$$$$$$$$$$$$$$$$$$$$$$$"
 70 LET A=USR 16514
 80 GOTO 70

500 FOR I=16514 TO 16536
510 PRINT I,PEEK I
520 NEXT I
```

Subroutine 500 geeft de van het vorige voorbeeld bekende methode om de opgeslagen codes op het scherm te zetten, ter kontrole van het programma.
Hoewel dit programma niet tot in alle bijzonderheden is verklaard, zal men het na een grondige studie toch wel kunnen bevatten.

NAWOORD

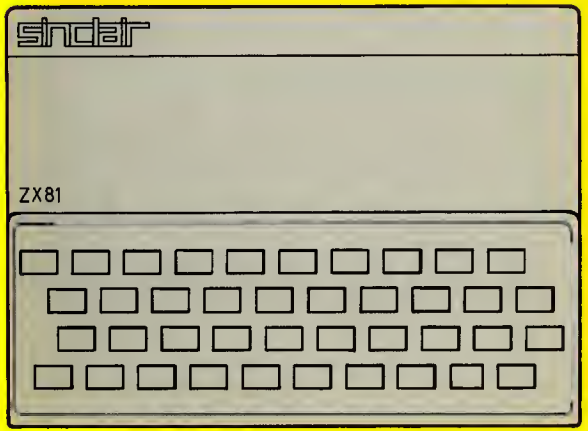
Hopelijk hebben de twee in hoofdstuk 10 behandelde voorbeelden van het programmeren in machinecode u zover gebracht dat uw aanvankelijke huiver voor het bestuderen van machinetaal is vervangen door de wens naar meer.
Zeker nu u met eigen ogen hebt gezien hoeveel sneller in machinecode geschreven programma's worden uitgevoerd, zal de volgende stap op de weg naar volleerd computerprogrammeur u voeren door het onontgonnen gebied van de machinetaal.
Er zijn een heleboel hoeken, die u daarbij kunt raadplegen.
Sommige zijn speciaal naar de ZX81 machine toegeschreven, andere behandelen de Z80-machinetaal in een meer algemeen, niet machinegebonden kader, maar zijn even bruikbaar.
Of u nu wel of niet besluit om over te stappen op het programmeren in machinetaal, wij hopen toch dat dit hoek u op weg heeft geholpen in het exploreren van de nieuwe mogelijkheden van de 16K-versie van de ZX81.
De mogelijkheden zijn eindeloos!

M.JAMES - S.M.GEE - ZX 81

ZX 81

LEREN PROGRAMMEREN

16K
16K
16K
16K
16K
16K
16K
16K



M.JAMES - S.M.GEE

DE MUIDERKRING